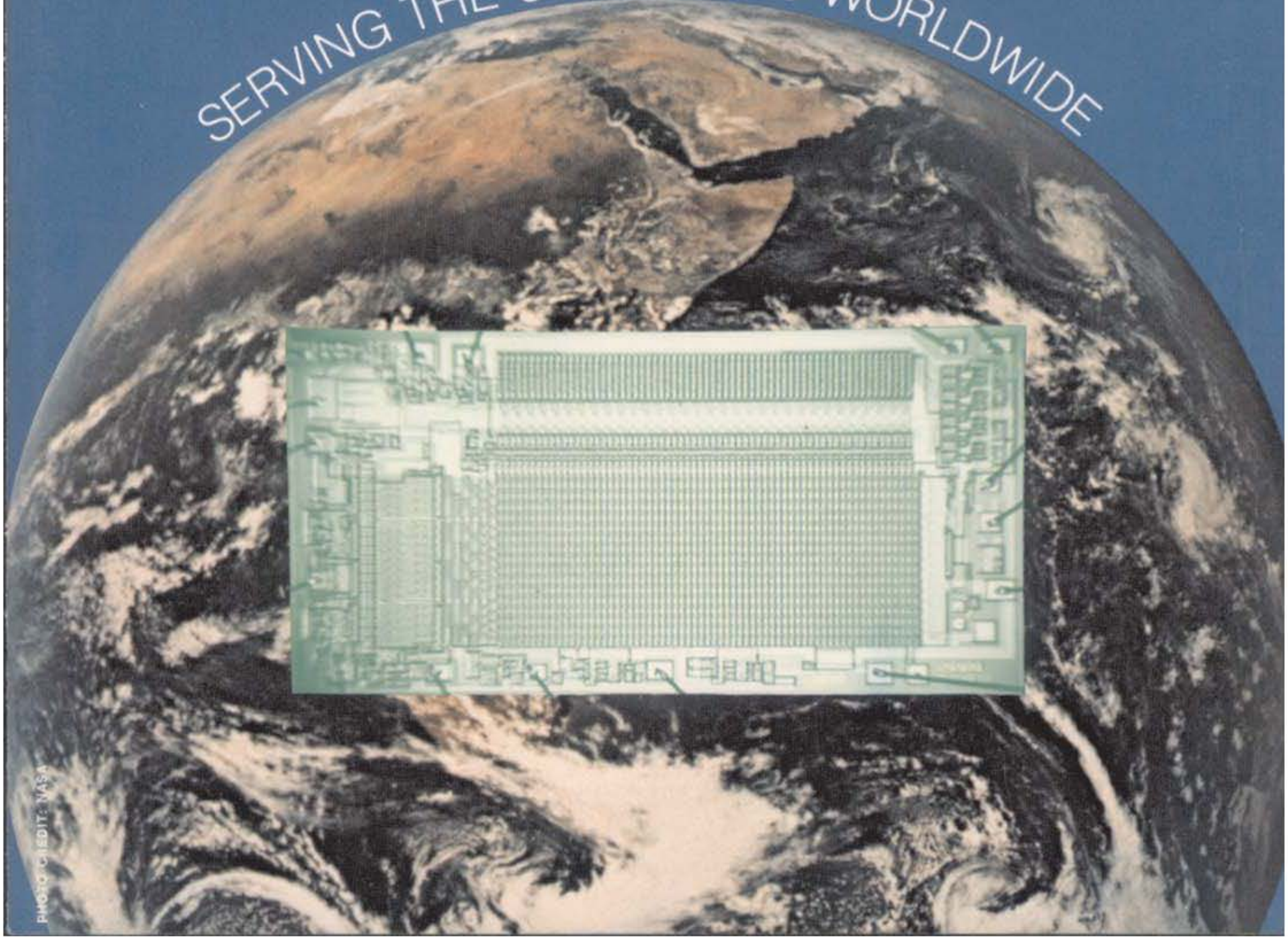
'68' MICRO JOURNAL

2.00

VOLUME I ISSUE 2 • Devoted to the 6800 User • March-April 1979

"Small Computers Doing Big Things."

SERVING THE 6800 USERS WORLDWIDE



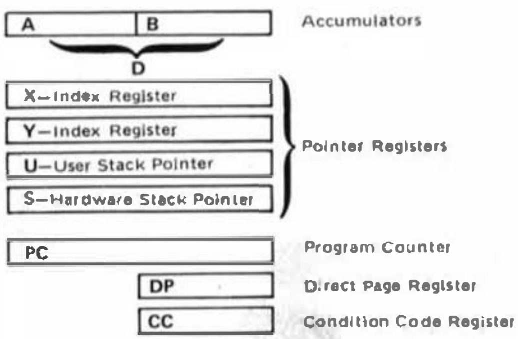
The World's Most Powerful 8-Bit Microcomputer

Featuring the World's Most Powerful 8-Bit MPU – The Motorola MC-6809

Welcome to a whole new world of microcomputing. Here at last is a microcomputer with all the speed and power that you have wished for. The MC6809 is an exciting new concept in microprocessors that fills the gap between 8- and 16-bit machines. It provides the power of 16-bit instructions with the economy of 8-bit architecture.

The MC6809 has more addressing modes than any other 8-bit processor. It has powerful 16-bit instructions, and a highly efficient internal architecture with 16-bit data paths. It is easily the most powerful, most software efficient, and the fastest 8-bit general purpose microprocessor ever.

6809 PROGRAMMING MODEL



The greatest impact of the Motorola MC6809 undoubtedly will be software related. Ten powerful addressing modes with 24 indexing sub-modes, 16-bit instructions and the consistent instruction set stimulate the use of modern programming techniques. Such as structured programming, position independent code, re-entrancy, recursion and multitasking.

A memory management system with extended addressing designed into the bus system controls up to 256K bytes of RAM memory. The dynamic memory allocation system, which is part of the multitasking DOS, allocates available memory in as small as 4K blocks.

The MC6809 system is the only 8-bit processor designed for the efficient handling of high-level languages. New addressing modes, a consistant instruction set and easy data manipulation on stacks allows the efficient execution of block-structured high-level code as generated by a compiler like PASCAL.

MP-09 Processor Card . . . .$ 195.00
68/09 Computer w/48K $1,500.00

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY
SAN ANTONIO, TEXAS 78216 (512) 344-0241

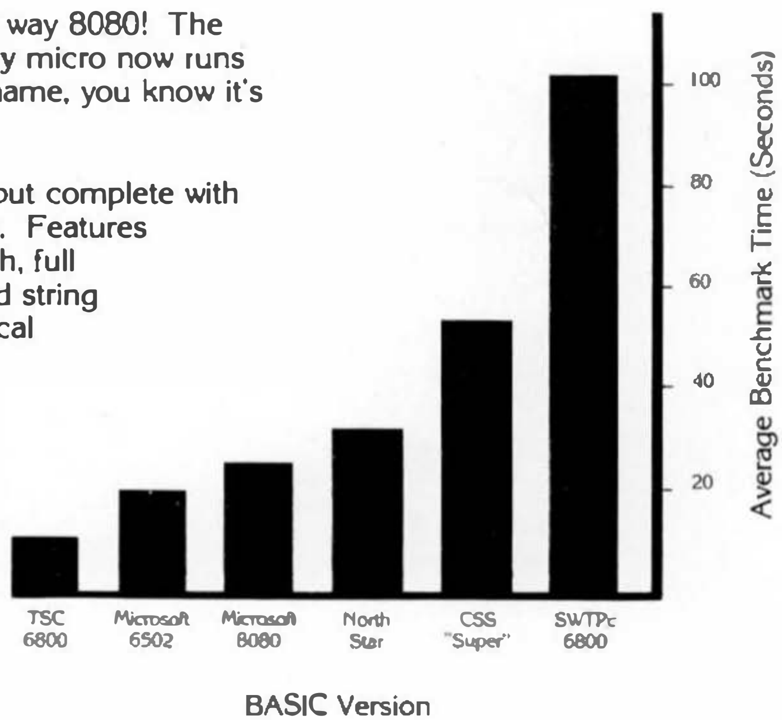
TSC BASIC for 6800

The fastest floating point BASIC for any micro.

Move over 6502! Out of the way 8080! The fastest floating point BASIC for any micro now runs on the 6800. And with the TSC name, you know it's top quality.

TSC BASIC is not only fast, but complete with over 50 commands and functions. Features include six digit floating point math, full transcendental functions, unlimited string length, if/then/else construct, logical operators,and two-dimensional arrays (including string arrays).

Available now on KCS cassette for $39.95. Requires 9K minimum, no source listing included. Soon to come is a version for the FLEX™ disk operating system.



Graph based on benchmarks listed in October 1977 issue of Kilobaud™ magazine.

Technical Systems Consultants, Inc.
Box 2574 W. Lafayette, IN 47906

Specialists In Software & Hardware for Industry & the Hobbyist

'68' MICRO JOURNAL

Portions of the text of '68' Micro Journal set using the following:
6800/2, DMAF1 and CT-82
Southwest Technical Products Corp.
219 W. Rhapsody
San Antonio, TX 78216

Editor, Word Processor and Sort/Merge
Technical Systems Consultants, Inc.
Box 2574
W. LaFayette, IN 47906

*MINIFLEX & FLEX REG ™ Technical Systems Consultants, Inc.

Selectric I/O
World Wide Electronics, Inc.
130 Northwestern Blvd.
Nashua, NH 03060

Publisher/Editor
Don Williams Sr.

Assistant Editor — Software
Mickey E. Ferguson

Assistant Editor — Hardware
Dennis Womack

Associate Editor — Southwest
Dr. Jack Bryant

Associate Editor — Midwest
Howard Berenbon

Subscriptions and Office Manager
Joyce Williams

Typography and Color Separations
Williams Company, Inc.
Chattanooga, TN 37421

CONTENTS

Send All Correspondence To:

'68' Micro Journal
3018 Hamill Rd.
PO Box 849
Hixson, Tennessee 37343

'68' Micro Journal is published 12 times a year by '68' Micro Journal, 6131 Airways Blvd., Chattanooga, TN 37421. Application to mail at second class postage rates is pending at Chattanooga, TN. Return postage guaranteed. Postmaster: Send Form 3579 to '68' Micro Journal, PO Box 849, Hixson, TN 37343.

Subscription Rates U.S.A.:
(Special Charter Rate)
1 year $10.50
2 years $18.50
3 years $26.50
Lifetime $125.00 (one-time payment) Twice rate shown above for Air Mail/First Class anywhere in the U.S.A.

—ITEMS SUBMITTED FOR PUBLICATION—

(Letters to the Editor for Publication) All 'letters to the Editor' should be substantiated by facts. Opinions should be indicated as such. All letters must be signed. We are interested in receiving letters that will benefit or alert our readers. Praise as well as gripes is always good subject matter. Your name may be withheld upon request. If you have had a good experience with a 6800 vendor please put it in a letter. If the experience was bad put that in a letter also. Remember, if you tell us who they are then it is only fair that your name 'not' be withheld. This means that all letters published, of a critical nature, cannot have a name withheld. We will attempt to publish 'verbatim' letters that are composed using 'good taste.' We reserve the right to define (for '68' Micro) what constitutes 'good taste.'

(Articles and items submitted for publication) Please, always include your full name, address, and telephone number. Date and number all sheets. TYPE them if you can, poorly handwritten copy is sometimes the difference between go, no-go. All items should be on 8X11 inch, white paper. Most all art work will be reproduced photographically, this includes all listings, diagrams and other non-text material. All typewritten copy should be done with a NEW RIBBON. All hand drawn art should be black on white paper. Please no hand written code items over 50 bytes. Neatly typed copy will be directly reproduced. Column width should be 3¼ inches.

(Advertising) Any Classified: Maximum 20 words. All single letters and/or numbers will be considered one (1) word. No Commercial or Business Type Classified advertising. Classified ads will be published in our standard format. Classified ads $7.50 one time run, paid in advance.

Commercial and/or Business advertisers please write or phone for current rate sheet and publication lag time.

CRUNCHERS CORNER

This monthly column is intended to provide a place for the exchange of ideas on microcomputer arithmetic. A systematic exposition of fixed and floating point arithmetic, hardware and software, algorithms for approximation and so on is planned. Questions and comments submitted to this column can be on any subject relevant to "number crunching," and should be addressed to:

Jack Bryant
Department of Mathematics
Texas A&M University
College Station, Texas 77843

We ask that all correspondents supply their names and addresses.

MULTIPLE PRECISION ARITHMETIC

In last month's column, we introduced 8 bit unsigned and two's complement representations and presented a simple program to let one examine how the addition instruction acted. We also gave an example of a modelling problem (exponential growth) in which the size of the numbers quickly exceeded 8 bits. One point of this example is the following: before an 8 bit processor can be used in most problems, it must be taught to handle much longer than 8 bit data types. This month, we deal with 16 bit integers.

The 6800 has several instructions which deal directly with 16 bit data. They involve the three 16 bit internal registers: the index register X, the stack pointer S and the program counter PC. While the stack pointer could possibly be used in some arithmetic operations, this use would compromise one powerful feature of the 6800 (the non-maskable interrupt). We will refrain from using the stack pointer for any other purpose. The program counter is not really available to the outside world. This leaves the index register X. Instructions which involve X are

DEX	Decrement Index Register
INX	Increment Index Register
TXS	Transfer Index Register to Stack Pointer
TSX	Transfer Stack Pointer to Index Register
CPX	Compare Index Register
LDX	Load Index Register
STX	Store Index Register

```
EXAMPLE OF USE OF INDEX REGISTER              '68 MICRO JOURNAL

0000 CE 00 00          LDX   #$0000  LOAD INDEX REGISTER WITH $0000.
0003 8C 27 10   LOOP   CPX   #10000  SEE IF THERE YET.
0006 27 03             BEQ   STOP
0008 08                INX           NOT THERE. BUMP INDEX REGISTER.
0009 20 F8             BRA   LOOP    DO IT AGAIN.
000B 3F         STOP   SWI           FINISHED--INTERRUPT PROCESSOR
                       END
     NO ERROR(S) DETECTED

  SYMBOL TABLE:
LOOP    0003        STOP    000B
```

Listing 1. Simple program to illustrate use of index register and storage order.

The first four of these instructions are single byte instructions which execute in 4 cycles. The next three have the same addressing modes as accumulator load, store and compare, and execute in only one more cycle than the corresponding accumulator A or B dual operand instructions. For example, the instruction sequence in Listing 1 executes the LOOP sequence 10000 times and then stops.

Note how the line labelled LOOP is translated:

```
0003 8C 27 10  LOOP  CPX  #10000
```

Note also that 10000_{10} = $2710. The two byte operand containing the 16 bit numbers to which the index register is being compared is stored high-then-low. That is, the most significant bits have lower addresses. In order to take advantage of these index register instructions in arithmetic operations, we must store the more significant byte first (with lower address). (It seems strange to store numbers otherwise, but other microprocessors require the reverse order.)

SIXTEEN BIT INTEGERS

Two byte (16 bit) integers will be interpreted as unsigned integers from 0 to 65535 or as two's complement integers from -32768 to 32767. We will always store them as suggested above: in two successive memory locations with the MSB first and the LSB second. This month's column contains an integer arithmetic package containing input (i.e. ASCII to binary), addition and subtraction. Arithmetic is carried out on (software) 16 bit registers I and M; I contains the result of an operation between I and M. The initial contents of I are replaced by the results of the operation.

(Thus I can be thought of as an accumulator.) For example, addition can be coded:

```
ADDI  LDA A I+1  FETCH LSB
      ADD A M+1  ADD
      STA A I+1  STORE
      LDA A I    FETCH MSB
      ADC A M    ADD WITH CARRY
      STA A I    STORE
```

The instruction ADC lets the computer perform a double precision addition. The carry bit is kept in the condition code register and is automatically added in when ACCA is added to the contents of location M. (Luckily, load and store instructions do not change the carry condition code.) Subtraction can be coded in essentially the same way:

```
SUBI  LDA A I+1  FETCH LSB
      SUB A M+1  SUBTRACT
      STA A I+1  STORE
      LDA A I    FETCH MSB
      SBC A M    SUBTRACT WITH BORROW
      STA A I    STORE
```

Instruction SBC performs a borrow from the LSB subtraction if bit C was set.

These routines each require 12 bytes and execute in 20 cycles; an additional 14 cycles is required to treat them as subroutines. There is an unexpected potential problem: the two's complement overflow condition is not always flagged correctly at the conclusion of ADDI or SUBI, although the carry (unsigned overflow) condition is. Also, the Z (zero result) bit is

set when the MSB of the result is zero. The N (negative) bit is correct.

CONVERSION

From last month's discussion we know how to convert from decimal to binary by hand. In this process, we know the decimal representation is a number. The computer knows a decimal number as a string of symbols. Conversions from decimal to binary in the computer are really conversions from the symbols (the minus sign and digits 0-9) to numbers. The symbols are encoded in ASCII, a 7-bit code. The minus sign, for example, is \$1D. The digits 0-9 are encoded \$30-\$39. It is thus easy to obtain the binary value of a digit (subtract \$30).

Once we have the digits, it is necessary to expand the representation. Recall (for example)

$$2743 = 2 \times 10^3 + 7 \times 10^2 + 4 \times 10 + 3 .$$

This expression appears to require a table of constants for conversion and several complicated multiplications. However, notice the expression is actually a polynomial in the number 10. The correct way to evaluate a polynomial is to nest it. (This is called Horner's method, although the method was first used by Newton.) Nesting gives

$$2743 = 3 + 10(4 + 10(7 + 10(2)));$$

while the same number of multiplications and additions are involved, only one constant is required: the constant 10. Multiplication by 10 is carried out by writing 10 = 8 + 2 ; multiplication by 8 is simply three left-shifts (with bit 0 being moved to the LSB). The following code multiplies the number in M, M+1 by 10 , leaving the LSB in ACCB and the MSB in ACCA.

```
LDA A M         ACCUMULATOR
LDA B M+1       SHIFTS ARE FASTER.
ASL B           ZERO INTO BIT 0
ROL A           *2
STA A M         SAVE TO
STA B M+1       ADD LATER
ASL B
ROL A           *4
ASL B
ROL A
ADD B M+1       ADD
ADC A M
 . . .
```

The result ends with the MSB in A and LSB in B .

The only remaining conversion problem is the sign problem. In the sign-magnitude representation, negative numbers in decimal are preceeded by a minus sign. In the program READI of Listing 2 , a flag is set if a leading minus sign is found, and the twos complement negative of the binary result of conversion is taken. If the MSB is in A and LSB is in B , this can be coded in 5 bytes as

```
      COM A         ONE'S COMPLEMENT MSB
      NEG B         TWO'S COMPLEMENT LSB
      BNE   SKIP    CARRY IF B IS ZERO
      INC A         ADD IN CARRY
SKIP . . .
```

In Flowchart I, the process outlined above is logically explained. Listing 2 subroutine READI performs this task. On entry, the B accumulator is saved. The index register points to the first byte of a string to be converted. On exit, the index register points to the next non-(0-9) character, and the number is in (M,M+1). No check on validity (such as too many decimal digits) is performed. Since this program is expected to advance the index register, it changes its value. On the other hand, the B accumulator is saved and restored so that it can have meaning outside the program. Of course, these restructions and conventions need to be made known to the user (that is, need to be documented); our first discussion of this general problem follows.

PROGRAMMING CONVENTIONS, STORAGE AND SPEED

When a large programming project is being planned, it is necessary to establish programming conventions. These are the groundrules which each acceptable subroutine must meet. The 6800 concept has several features which let programs be almost "universal." For example, it is possible to write reentrant code which can be shared by several users; parameters are passed to the program through the stack or accumulators. Such programs generally use indexed addressing, which is slow. On the other hand, it is also possible to write programs which execute very rapidly. This is done using the direct (page zero absolute) addressing mode, which of course is not reentrant. The philosophy followed here places roughly the same premium on storage and speed, but does not aim for reentrant code.

The 256 bytes on page zero need to be managed carefully if many programs using memory here are to fit together. READI , for example, uses 4 bytes: (M, M+1) is a program variable. Two other temporary variables are required (ATEMP and NEGFLG); these page zero locations are available for use outside READI , but it must be understood that READI will change their contents. (M,M+1) is changed, but this is expected.

The groundrules followed here for all programs intended to exist as a "package" to be used by other programs are simple:

1. The A accumulator is purely local. That is, any subroutine may modify ACCA without warning.
2. The B accumulator, however, is global: a subroutine which changes ACCB must restore the original value before returning. The exceptions to this are obvious: B could be an input parameter (such as a counter which is expected to be decremented until its value is zero), or could be used to contain an output parameter. These exceptions must be carefully documented.
3. The index register X must similarly be global, with the same exceptions as apply to ACCB . This leads to an interesting memory management problem since no construct analogous to PSH B/PUL B exists for the index register. The code

```
ENTRY PSH B           SAVE ACCB ON STACK
      STX   XTMP      PREPARE TO SAVE X
      LDA B XTMP
      PSH B           SAVE X
      LDA B XTMP+1
      PSH B
      . . .
EXIT  PUL B
      STA B XTMP+1
```

```
PUL B            RESTORE X
STA B XTMP
LDX   XTMP
PUL B            RESTORE ACCB
RTS
```

simulates the convention required, but is lengthly and slow. (A PSX/PUX instruction pair would be dandy.) Of course this is only required for programs which might (directly or indirectly) call themselves. The memory management problem can be solved by setting aside page zero temporaries for each "package" of subroutines.

4. Each program must consist of code which executes in read only memory (ROM). (By using temporary random access memory (RAM) to execute very short programs with self-modifying code, we can retain this useful feature.)
5. Each program must tolerate the non-maskable interupt instruction. In particular, the stack pointer is not to be used as an index register.

INPUT/OUTPUT

Subroutine READI of Listing 2 expects a decimal integer in ASCII to be pointed to by the index register. If one is found, it is translated and placed in M . Control returns to the calling program with the index register pointing to the next character which is not a digit 0-9.

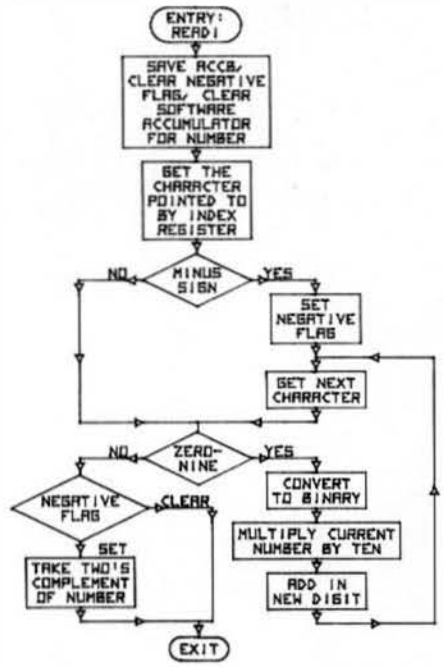
RAM	BEFORE	AFTER
0100	INDEX → -	-
0101	1	1
0102	0	0
0103	4	4
0104	9	9
0105	+	INDEX → +
0106	2	2
0107	1	1
0108	0	0

Subroutine INPUT in Listing 2 performs the simple task of transferring numbers and operands from the computer console to the input buffer. Only checking for a "return" (ASCII $0D) is performed; this is echoed by a linefeed and control is returned to the calling program. (The input buffer is always accessed using indexed addressing and accordingly need not be given page zero memory space.)

Subroutine OUTPUT prints the result of the computation in hexadecimal. Next month, we discuss binary to signed decimal ASCII conversions. For the time being, we simply print the value of I in hex. This is useful since we also get a feeling for the hex representation.

The main program in Listing 2 is the subroutine BANG . This routine scans the input buffer looking for a sequence operand, operator, operand, ... The operands are converted to 16 bit binary with subroutine READI. Allowable operators are + and - . The line is assumed to be terminated by a carriage return. When an illegal character is found, a '?' is printed on the system console and control is returned to the calling program.

In next month's column, we discuss multiplication, division and conversion from binary to ASCII . The program in Listing 2 will be expanded to a 16 bit "four banger."



FLOW CHART 1: CONVERSION ASCII —> BINARY

Listing 1. Integer Arithmetic Package, Version 0.1

The main features of the package as developed so far are conversion from ASCII to binary, addition and subtraction, and a program to interpret (left-to-right) a string of operands (numbers) and operators (+, - and RETURN) entered from the system console.

```
INTEGER APITHMETIC PACKAGE  V0.1                '68 MICPO JOUPNAL      PAGE 1

                       *        MINBUG EQUATES
                       *
E07E                   PDATA1 EQU  $E07E   BLOCK OUTPUT
E1AC                   INEEE  EQU  $E1AC   INPUT ONE CHAPACTEP
E1D1                   OUTEEE EQU  $E1D1   CHAPACTER OUTPUT POUTINE
E0C8                   OUT4HS EQU  $E0C8   HEX OUTPUT POUTINE: TWO BYTES
                       *
                       *        PAGE ZEPO CONSTANTS
                       *
00F0                          OPG  $F0
00F0                   ATEMP  PMB  1       TEMPOPARY ACCUMULATOP STOPE
00F1                   INBUF  PMB  2       INPUT BUFFEP BEGINNING
00F3                   NEGFLG PMB  1       NEGATIVE INPUT FLAG
00F4                   M      PMB  2       M SOFTWAPE ACCUMULATOP
00F6                   I      PMB  2       I SOFTWAPE ACCUMULATOP
00F8                   XTEMP  PMB  2       TEMPOPARY STOPAGE FOP INDEX PEGISTEP
                       *
                       *        INDEXED PEGISTEP ADDPESSED STOPAGE ON PAGE 1
                       *
0200                          OPG  $0200   TEST POUTINE STAPTS HEPE
                       *
                       *        TEST PPOGPAM FOP INTEGEP APITHMETIC PACKAGE
                       *             VEPSION 0.1
                       *
0200 CE 02 CE                 LDX  #CPLFD
0203 BD E0 7E                 JSP  PDATA1  START WITH A NEW LINE.
0206 8D 08             TEST   BSP  INPUT    GET NUMBEP AND OPEPATOP STPING
0208 DE F1                    LDX  INBUF
020A 8D 3A                    BSP  BANG     PEPFOPM OPEPATIONS
020C 8D 27                    BSP  OUTPUT   PPINT THE PESULT
020E 20 F6                    BPA  TEST     DO IT AGAIN
                       *
                       *
                       *        INPUT SUBPOUTINE
                       *
                       *             SUBPOUTINE TO TAKE A STPING FROM THE CONSOLE
                       *             AND STOPE IN INPUT BUFFEP.
                       *
0210 DF F8             INPUT  STX  XTEMP   SAVE INDEX PEGISTEP
0212 37                       PSH B        SAVE ACCB
0213 5F                       CLP B        SET COUNTEP
0214 CE 01 00                 LDX  #$100   INPUT BUFFEP STAPTS AT $100
0217 DF F1                    STX  INBUF
0219 BD E1 AC          INEXT  JSP  INEEE   GET A CHAPACTEP
021C A7 00             ISTASH STA A 0,X    STOPE IT
021E 08                       INX
```

```
021F 5C               INC B             BUMP COUNTER
0220 29 0E            BVS    IFULL      BUFFER SIZE = 127
0222 81 0D            CMP A #$D         CR?

INTEGER ARITHMETIC PACKAGE  V0.1              '68 MICRO JOURNAL

0224 26 F3            BNE    INEXT      NOPE
0226 CE 02 CF  IENTRY LDX    #CRLFD     LOAD DATA FOR CR/LF
0229 BD E0 7E         JSR    PDATA1
022C DE F8            LDX    XTEMP      RESTORE INDEX REGISTER
022E 33               PUL B             RESTORE ACCB
022F 39               RTS
0230 09        IFULL  DEX               BUFFER FULL: GENERATE CR.
0231 86 0D            LDA A #$D
0233 20 E7            BRA    ISTASH
               *
               *      OUTPUT SUBROUTINE
               *
               *             THIS PROGRAM OUTPUTS I IN HEX.
               *
0235 DF F8     OUTPUT STX    XTEMP      STASH X
0237 CE 00 F6         LDX    #I         LOAD WITH ADDRESS OF I
023A BD E0 C8         JSR    OUT4HS     PRINT IT
023D CE 02 CE         LDX    #CRLFD     PRINT A CR/LF
0240 BD E0 7E         JSR    PDATA1
0243 DE F8            LDX    XTEMP      RESTORE X
0245 39               RTS
               *
               *      SUBROUTINE BANG
               *
               *  THIS SUBROUTINE TAKES THE STRING IN THE INPUT
               *  BUFFER AND LOCATES THE OPERATORS AND CONSTANTS
               *  AND CALLS THE APPROPRIATE ROUTINE TO APPLY THE
               *  OPERATION.
               *
               *             OPERATIONS SO FAR:
               *
               *      +      ADDITION:       I := I + M
               *      -      SUBTRACTION:    I := I - M
               *
0246 37        BANG   PSH B
0247 8D 28            BSR    READI      CONVERT ONE NUMBER
0249 96 F4            LDA A M           SAVE THE
024B 97 F6            STA A I           FIRST ONE
024D 96 F5            LDA A M+1         IN I
024F 97 F7            STA A I+1         TOO.
0251 E6 00     BANGN  LDA B 0,X         LOAD THE OPERATOR
0253 08               INX               POINT TO NEXT OPERAND
0254 C1 20            CMP B #$20
0256 2D 17            BLT    OVER       SEE IF A CONTROL CHARACTER
0258 8D 17            BSR    READI      GET THE NEXT OPERAND
025A C1 2B            CMP B #'+
025C 26 04            BNE    SUBO
025E 8D 54            BSR    ADDI       ADD
0260 20 EF            BRA    BANGN
0262 C1 2D     SUBO   CMP B #'-

INTEGER ARITHMETIC PACKAGE  V0.1              '68 MICRO JOURNAL

0264 26 04            BNE    ERROR
0266 8D 59            BSR    SUBI       SUBTRACT
0268 20 E7            BRA    BANGN      DO IT AGAIN
026A 86 3F     ERROR  LDA A #'?         LOAD AN ERROR FLAG.
026C BD E1 D1         JSR    OUTEEE     PRINT IT
026F 33        OVER   PUL B             RESTORE ACCB
0270 39               RTS
               *
               *      READ SUBROUTINE
               *
               *  ENTER WITH INDEX REGISTER POINTING TO STRING
               *  TO BE CONVERTED TO 16 BIT BINARY.
               *
               *  EXIT WITH INDEX REGISTER POINTING TO NEXT NON
               *  (0-9) OR LEADING MINUS SIGN AND WITH NUMBER IN M.
               *
0271 37        READI  PSH B             SAVE ACCB
0272 5F               CLR B             CLEAR ACCUMULATOR
0273 37               PSH B
0274 D7 F3            STA B NEGFLG      CLEAR NEGATIVE FLAG
0276 A6 00            LDA A 0,X         GET DIGIT/SIGN
0278 81 2D            CMP A #'-         LEADING MINUS?
027A 26 05            BNE    GOON       NO
027C 97 F3            STA A NEGFLG      YES
027E 08        NEXT   INX               PREPARE TO GET NEXT ONE.
027F A6 00            LDA A 0,X         DO IT
0281 81 30     GOON   CMP A #'0         >= ZERO?
0283 2D 1E            BLT    NOTONE
0285 81 39            CMP A #'9         <= NINE?
0287 2E 1A            BGT    NOTONE
0289 80 30            SUB A #'0         GOT ONE, CONVERT TO BINARY
028B 97 F0            STA A ATEMP       SAVE FOR LATER
028D 32               PUL A             RESTORE M:B
028E 58               ASL B             *
028F 49               ROL A             *
0290 97 F4            STA A M           *
0292 D7 F5            STA B M+1         *      MULTIPLY BY 10
0294 58               ASL B             *
0295 49               ROL A             *
0296 58               ASL B             *
0297 49               ROL A             *
0298 DB F5            ADD B M+1         *
029A 99 F4            ADC A M           *
029C DB F0            ADD B ATEMP       ADD IN DIGIT
029E 89 00            ADC A #0          AND CARRY OUT.
02A0 36               PSH A             FREE ACCA
02A1 20 DB            BRA    NEXT       DO IT AGAIN
02A3 32        NOTONE PUL A             HERE WE DIDN'T FIND A DIGIT.
02A4 7D 00 F3         TST    NEGFLG     NEGATIVE?
02A7 27 05            BEQ    SKIP

INTEGER ARITHMETIC PACKAGE  V0.1              '68 MICRO JOURNAL

02A9 43               COM A
02AA 50               NEG B             MAKE IT NEGATIVE.
02AB 26 01            BNE    SKIP
02AD 4C               INC A             CARRY OUT
02AE 97 F4     SKIP   STA A M           STASH THE NUMBER
02B0 D7 F5            STA B M+1         WE BUILT.
02B2 33               PUL B             RESTORE ACCB
02B3 39               RTS
               *
               *      SUBROUTINE ADDI
               *
               *ADDS M TO I AND PLACES THE RESULT IN I.
               *
02B4 96 F7     ADDI   LDA A I+1
02B6 9B F5            ADD A M+1
02B8 97 F7            STA A I+1
02BA 96 F6            LDA A I
02BC 99 F4            ADC A M
02BE 97 F6            STA A I
02C0 39               RTS
               *
               *      SUBROUTINE SUBI
               *
               * SUBROUTINE TO SUBTRACT CONTENTS OF M FROM I AND
               * PLACE IN I.
               *
02C1 96 F7     SUBI   LDA A I+1
02C3 90 F5            SUB A M+1
02C5 97 F7            STA A I+1
02C7 96 F6            LDA A I
02C9 92 F4            SBC A M
02CB 97 F6            STA A I
02CD 39               RTS
               *
02CE 0D        CRLFD  FCB    $D,$A,4   DATA FOR CR/LF
02CF 0A 04
                      END
        NO ERROR(S) DETECTED
```

SYMBOL TABLE:

ADDI	02B4	ATEMP	00F0	BANG	0246	BANGN	025
CRLFD	02CE	ERROR	026A	GOON	0281	I	00F
IENTRY	0226	IFULL	0230	INBUF	00F1	INEEE	E1F
INEXT	0219	INPUT	0210	ISTASH	021C	M	00F
NEGFLG	00F3	NEXT	027E	NOTONE	02A3	OUT4HS	E0C
OUTEEE	E1D1	OUTPUT	0235	OVER	026F	PDATA1	E07
READI	0271	SKIP	02AE	SUBI	02C1	SUBO	02e
TEST	0206	XTEMP	00F8				

A Look at the SWTPC CT-82 Video Terminal

Mickey E. Ferguson
POB 708
Trenton, GA 30752

You probably know Southwest Technical Products Corporation best as a manufacturer of 6800 based computer systems. But they were in the computer terminal business for quite some time before they started making computers. SWTPC's first terminal was the TVT-1. The TVT-II (or CT-1024, if you prefer) was their next offering and was the most popular terminal with computer hobbyists in it's day. The 1024, as it was most

often called by hobbyists, was restricted to only thirty-two upper case characters per line, sixteen lines per screen. and a maximum baud rate of 1200 bps. It also had no chassis or case. In it's favor, the 1024 was economical (about $300) and it worked and worked and worked. As a result a "cottage industry" developed, offering after-market modifications to add scrolling, sixty-four character line length, lower case, etc. Then SWTPC replaced the 1024 with the CT-64; to calm the restless natives it offered most of the features that CT-1024 owners had been adding to their terminals. In addition, the CT-64 came with an excellent Motorola video monitor as well as a chassis and cover.

A REAL terminal is here now. It is the CT-82, and it sets new standards for video terminals. End of history lesson.

Forget everything that you may know about video terminals because it no longer applies. The CT-82 is the first of an entirely new generation of video terminals. It is built around a 6802 microprocessor and a 6845 CRT controller chip. A what? A CRT controller chip (CRTC), of course. According to Motorola, "The MC6845 CRT Controller performs the interface to raster scan CRT displays. It is intended for use in processor-based controllers for CRT terminals." The CRTC takes care of the video generation, video timing, etc.; while the 6802 communicates with you thru the keyboard, with the computer thru an RS-232 interface, and (optionally) with a printer thru a parallel interface. The CRTC and the 6802 communicate with each other and with memory via a common address and data bus. So the CRTC and the microprocessor work together as a stand alone memory mapped video display. Since the 6802 works in memory during one phase of the clock, while the CRTC is in memory during the other phase; each is able to operate at maximum speed. Neither device slows the operation of the other and each needs never be aware of the fact the other device ever has control of the bus. The end result, the CT-82, is a terminal that costs less and does more than seems reasonably possible. But, is the CT-82 really a terminal? Isn't it really a microcomputer that simply thinks it is a terminal? Whatever the CT-82 may actually be inside, it is a quality video terminal at a reasonable price.

First Impressions

I had difficulty believing the little box that the UPS man brought actually contained my CT-82. How could a container so small have a CRT terminal inside? Upon lifting the box, I was completely convinced that a horrible mistake had been made. It did not weigh enough to have a complete CRT terminal in it. But it did. The CT-82 only weighs about twenty pounds (9 kg) and measures ten inches (25.4 cm) high, by eighteen inches (45.7 cm) wide, by seventeen inches (43.2 cm) deep.

SWTPC has been advertising the CT-82 just about everywhere except in porno magazines, so you must have seen its picture. The pictures do not do it justice. The CT-82 is cute (my wife says it's darling) unlike its predecessor, the CT-64. It blends well into almost any home or office environment without attracting undue attention to itself or creating an eyesore.

The nine inch (diagonally measured) CRT caused some disappointment on my part initially. But this was countered by the "senuous feel of the keyboard" (to quote John Craig, Editor of Creative Computing). The keyboard uses Cherry keyswitches for both high reliability and excellent feel. There is also a twelve key pad for cursor control and editing functions in addition to the regular keyboard. Optionally, the cursor control pad can be replaced with a numeric pad should you desire it enough to pay extra for the added convenience. Other than the keyboard, the only external controls are the power switch and the configure switch. There is a DB-25 female RS232 connector on the rear of the CT-82 for communication with the outside world.

The configure switch allows the user to tell the CT-82 what is expected of it when power is first applied. It has two positions, which are auto and programmable. With the configure switch in the auto position, the CT-82 assumes it is to operate in the conversational mode, in full duplex, with sixteen lines per screen, at 9600 baud. With the configure switch in the programmable position, the power up options are controlled by an internal dip switch. The dip switch allows the user to select either conversational or paged edit modes, half

generator installed, in the graphics mode the CT-82 displays twenty two lines of ninety two characters. The upper case character set is the standard ASCII upper case, while the lower case is composed of graphic symbols. For graphics purposes, each character position is divided into six pixels (two horizontally by three vertically). This gives a resolution of one hundred eighty four pixels horizontally and sixty six pixels vertically. The limitation on graphics resolution in the standard CT-82 is the amount of memory in the terminal. I will not be at all surprised when some enterprising individual devises a simple modification to expand the graphics resolution of the CT-82 to five hundred twelve horizontal by two hundred fifty six vertical.

The graphics character set in SWTPC's character generator, in my opinion, leaves much to be desired. When I am attempting to use the graphic character set, I invariably need some symbol that is not there. However, this does not present as much of a problem as you might think because you do not have to use the graphics character set. You see, the CT-82 can also do co-ordinate graphics (like the TRS-80). In fact, any graphics program written for the TRS-80 can be easily modified to run on a computer using the CT-82. The biggest difference between graphics on the TRS-80 and the CT-82 is that the CT-82 can do more with less programming. To draw a graphics line on the TRS-80, you must turn on each pixel individually. If you know the starting and ending points of the line, the CT-82 can draw the entire line for you with a single command from the computer. The CT-82 can also invert your graphics allowing white on black or black on white. Careful use of the CT-82's Yaw, Pitch, Slide, Roll, Scroll, and Invert commands lets you approach animation. This is particularly impressive at 38.4k baud.

As mentioned earlier, SWTPC is offering a companion editor for use with 6800 systems utilizing FLEX and a CT-82. It is the first piece of software offered by SWTPC that was written by SWTPC, and they "modestly" refer to it as THE EDITOR. Thru the efforts of Dan Meyer at SWTPC, I was able to get an early copy of their Editor for evaluation.

The Editor works from input file to memory to output file; it is able to make multiple passes thru the file without returning to the operating system. As a new pass is begun, the old output file becomes the new input file and the old input file becomes the new output file. The Editor uses two different types of commands. There are alphabetic commands like D - delete, I - insert, and F - find. There are also commands that are entered from the cursor control pad. For example, pressing the FORM key on the cursor control pad moves the text on the screen up nineteen lines or one page. A puzzling feature of the editor is its ability of text manipulation by line number. This is interesting because it does not use nor display line numbers.

The choice and evaluation of text editors is a very subjective business. A lot depends upon the type of work done with the editor. Everyone seems to have his or her personal favorite. I must admit that I prefer the TSC editor to the SWTPC editor. Don Williams takes the opposing view preferring the SWTPC editor. *(Ed's Note: I actually prefer the TSC Editor for some functions and the SWTPC Editor for others. An example is our mailing list; the SWTPC Editor is much simpler to use. For text, such as this article I prefer the TSC Editor.)*

The CT-82 is indeed an impressive terminal, but has perfection been achieved? I scarcely thought so, at first. The 9" monitor just had to be too small. After living with it for a while my opinion has changed. The 12" monitor on my old terminal seems too big. One does not look at the CRT on a terminal from across the room and at normal viewing distances, the 9" CRT is quite adequate. This is perhaps enhanced by its extreme sharpness and clarity. It is rather unusual for a CRT to be focused properly at all points on the screen. Often the edges are fuzzy when the center is sharp, and vice versa. Such is not the case with the CT-82, the display is very sharply focused at all points on the CRT screen. All of the CT-82's I have seen have shared this characteristic, which causes me to believe that it is not unique to my terminal. I have acquired the habit of using the graphics character generator (with the 92 by 22 screen format) any time that I do not need lower case. This

generator installed, in the graphics mode the CT-82 displays twenty two lines of ninety two characters. The upper case character set is the standard ASCII upper case, while the lower case is composed of graphic symbols. For graphics purposes, each character position is divided into six pixels (two horizontally by three vertically). This gives a resolution of one hundred eighty four pixels horizontally and sixty six pixels vertically. The limitation on graphics resolution in the standard CT-82 is the amount of memory in the terminal. I will not be at all surprised when some enterprising individual devises a simple modification to expand the graphics resolution of the CT-82 to five hundred twelve horizontal by two hundred fifty six vertical.

The graphics character set in SWTPC's character generator, in my opinion, leaves much to be desired. When I am attempting to use the graphic character set, I invariably need some symbol that is not there. However, this does not present as much of a problem as you might think because you do not have to use the graphics character set. You see, the CT-82 can also do co-ordinate graphics (like the TRS-80). In fact, any graphics program written for the TRS-80 can be easily modified to run on a computer using the CT-82. The biggest difference between graphics on the TRS-80 and the CT-82 is that the CT-82 can do more with less programming. To draw a graphics line on the TRS-80, you must turn on each pixel individually. If you know the starting and ending points of the line, the CT-82 can draw the entire line for you with a single command from the computer. The CT-82 can also invert your graphics allowing white on black or black on white. Careful use of the CT-82's Yaw, Pitch, Slide, Roll, Scroll, and Invert commands lets you approach animation. This is particularly impressive at 38.4k baud.

As mentioned earlier, SWTPC is offering a companion editor for use with 6800 systems utilizing FLEX and a CT-82. It is the first piece of software offered by SWTPC that was written by SWTPC, and they "modestly" refer to it as THE EDITOR. Thru the efforts of Dan Meyer at SWTPC, I was able to get an early copy of their Editor for evaluation.

The Editor works from input file to memory to output file; it is able to make multiple passes thru the file without returning to the operating system. As a new pass is begun, the old output file becomes the new input file and the old input file becomes the new output file. The Editor uses two different types of commands. There are alphabetic commands like D - delete, I - insert, and F - find. There are also commands that are entered from the cursor control pad. For example, pressing the FORM key on the cursor control pad moves the text on the screen up nineteen lines or one page. A puzzling feature of the editor is its ability of text manipulation by line number. This is interesting because it does not use nor display line numbers.

The choice and evaluation of text editors is a very subjective business. A lot depends upon the type of work done with the editor. Everyone seems to have his or her personal favorite. I must admit that I prefer the TSC editor to the SWTPC editor. Don Williams takes the opposing view preferring the SWTPC editor. *(Ed's Note: I actually prefer the TSC Editor for some functions and the SWTPC Editor for others. An example is our mailing list; the SWTPC Editor is much simpler to use. For text, such as this article I prefer the TSC Editor.)*

The CT-82 is indeed an impressive terminal, but has perfection been achieved? I scarcely thought so, at first. The 9" monitor just had to be too small. After living with it for a while my opinion has changed. The 12" monitor on my old terminal seems too big. One does not look at the CRT on a terminal from across the room and at normal viewing distances, the 9" CRT is quite adequate. This is perhaps enhanced by its extreme sharpness and clarity. It is rather unusual for a CRT to be focused properly at all points on the screen. Often the edges are fuzzy when the center is sharp, and vice versa. Such is not the case with the CT-82, the display is very sharply focused at all points on the CRT screen. All of the CT-82's I have seen have shared this characteristic, which causes me to believe that it is not unique to my terminal. I have acquired the habit of using the graphics character generator (with the 92 by 22 screen format) any time that I do not need lower case. This

format gives characters about the same size as an office typewriter and puts more information on the screen at one time. Now the 9" monitor seems too big when I have to use either of the lower case screen formats.

I nearly threw that "sensuous" keyboard in the garbage. I thought SWTPC had done it again. Keys that felt like mush, keys that didn't do anything, and most of the keys were double striking. However the mushy keys and the non-working keys were the fault of the shippers. And the double striking keys were caused by a capacitor of the wrong value not by the key switches. It seems that some early CT-82's (mine included) had a .001 mfd instead of a .01 mfd capacitor for C8 on the keyboard. If you have an early CT-82 with double striking keys, I urge you to check the value of this capacitor.

My biggest complaint (and only really valid one) is the lack of a RESET button. I'm not the best programmer around and my programs often run away printing wildly. A run away program cannot suppress the printing of control characters. Many strange control character sequences are printed which configure the CT-82 in ways it was never intended to be configured. The only way to recover is to turn the power off on the CT-82. I need a means of recovery such as a RESET button. Or some way to return to the configuration established when the CT-82 is turned on, short of turning power off.

Conclusion

The CT-82 is one of the brightest of the smart terminals available today and truely a pleasure to use. For the approximate cost of a dumb terminal ($795) it is probably today's best buy in CRT terminals. It offers many features unavailable on terminals costing three times as much. Unlike the earlier offerings from SWTPC, the CT-82 is truely a professional quality terminal. But like all SWTPC products, it has a "hobbyist quality" price.

** SUBSCRIPTION NOTICE **

As most readers are aware, 68 Micro Journal came out approximately one month late. This was a direct result of a paper order mistake that was beyond our control. As a result we ended up placing the magazine, in the hands of the postal service, near the end of the monthly issue date. Most February copies were received in March. Issue number 1 was dated February 1979 and mailed the latter part of February. It should have been mailed the latter part of January 1979. So, we were a month late.

In order that we may get back to a more normal schedule, this issue is dated March-April 1979. The May issue then can be mailed at a more reasonable time; latter part of April. This will in no way cause any subscriber to miss a single issue of 68 Micro Journal. It simply means that if your subscription was due to expire in January 1980, it will expire with the February issue 1980. So you see, you will still have received 12 seperate issues, one each month, provided yours is a one year subscription. Of course if you are subscribed for two or three years the same hold true, for the one month extension. Life subscribers of course really don't care, that is just as long as they receive 68 Micro Journal each month, till then. They will.

LETTERS

Tom Harmon
Box 493
Laurel,Md.
20810
To: Don Williams

Dear Mr. Williams,

A comment about your FLEX TO BFD article; This is the kind of article that I think that most of the readers enjoy but, I must pick at the reasons given by Mr. Puckett for the conversion to FLEX. His comment that the TTYSET utility is an overpowering reason to change to FLEX is lost on me. I also have written a conversion and ran FLEX only to find out that when I set the TTYSET for my video terminal (16 lines and then stop) that the damn thing also stopped in the middle of a BASIC programme. BASIC uses the disk IO for its' IO and this is very disturbing.

I also found the forced extensions a total mess as in the SMOKE DOS I have about 10 megabytes of software and cannot

afford to have TXT after every text file.

VERY IMPORTANT is the fact that my editor and my word processor and my BASIC int. as well as compiler can read and work on ANY file of ANY kind when there are NO forced extensions.

I find this and the almost unlimited utilities available alone to set the SMOKE DOS so much ahead of FLEX that I can't find anyone in this area who uses flex anymore. By the way, SMOKE sells a board for SWTP disk users that will upgrade them in hardware and software to the SMOKE System.

Don't forget that on the SMOKE I have: Several BASIC's, (2 are compilers), Fortran, Pilot, Fourth, Random access (for a while now), and the detail that ALL software used on SMOKE DOS 'MON68' (the first), thru DOS4.2, will run on any other version with no changes. That is software! Please don't forget that all of this is compatible with existing hardware and software. Have any of you tried running FLEX 8" with a 32k board? FLEX 5 and FLEX 8 cannot copy a programme from one to the other without a lot of hassle. On my SMOKE I run 2ea 8" drives mixed with 2ea 5" drives, AND THE SOFTWARE DOES'T CARE!!

All in all, I enjoyed the article, but you can see I am very opinionated. Thanks for the great magazine!

Tom Harmon

COMPUTER MART

March 6, 1979

Don Williams Sr.
68 Micro Journal
3018 Hamill Road
P.O. Box 849
Hixson, Tenn. 37343

Dear Don,

Congratulations on a fine first issue. We will be proud to sell the 68 Micro Journal and advertise in it too.

One thing though, in the article by our old friend Dale Puckett, "FLEX to BFD", he says that the Computer Mart of New York told him that the Smoke Signal System was the only one to consider because of the poor software delivered with the SWTPC MP-68. That was correct at the time, because South West was supplying a disk operating system called FDOS with the MP-68. We considered that to be inferior to the BFD operating system and I suppose that SWTPC did also because they changed to FLEX.

As one of the oldest computer stores selling 6800 equipment, we are dedicated to providing support to 6800 users. We have sold SWTPC, MSI, Smoke Signal equipment for three years now and 6800 users are our most loyal customers.

It is about time that there was a publication devoted to the most reliable and interesting of all computer systems.

Sincerely,

Stan Veit
Storekeeper

P.S. We also support GMIX, TSC and Microware products, they are all great.

COMPUTER MART OF NEW YORK, INC.
118 [illegible] Avenue • New York, New York 10016 • [illegible]

Howard Berenbon
27200 Franklin Rd. #105
Southfield, MI 48034

6800 RELATIVE MODE BRANCHING, BY HAND

Here's a simple method for the machine language programmer to aid in calculating the values of relative mode branches with the 6800. It's called "COUNT".

Using "COUNT" eliminates the need to derive the branch value with two's-compliment arithmetic, thus eliminating a tedious chore. I use a subtraction method to arrive at the required value, quickly.

"COUNT" allows calculating values for 'forward' and 'backward' relative branches. The method for calculating these branch values are not identical, but similar. Although the examples given for forward and backward "COUNT" are very short branch distances, these rules may be applied successfully to branches whose destinations are + and - 128 (decimal).

"COUNT" is quick and simple to use. It eliminates using two's compliment arithmetic and save time for the machine language programmers. (Ed's note: Bless their hearts, long may they live and code.)

Forward Branching

Forward branching is somewhat simpler than backward, requiring only a value (in hex) as the distance from the address of the branch statement to it's forward destination, minus 02. In other words,

subtract the address of the branch statement from the address of the branch destination. Then subtract 02 from the value (see example below). Destination address - Branch address - 02 = Branch value:

Example:

```
Address     Object Code    Mnemonic

1000        B6 1020 SUB    LDAA NUMBER    100B
1003        26 03          BNE  DEST     -1003
1005        C6 09          LDAB 09   ---------
1007        39             RTS              05
1008        4A      DEST   DECA            -02
1009        B7 1020        STAA NUMBER --------
100C        7E 1000        JMP  SUB         03
```

Backward Branching

Backward "COUNT" is calculated by subtracting the backward destination address from the branch statement address, and a 01 is added to the result. Then, the final value is subtracted from FF (hex) to arrive at the branch value.

FF = ((Branch address - Destination address) + 01) = Brackward Branch Value:

Example:

```
Address     Object Code    Mnemonic

1000        B6 1020  UP    LDAA NUMBER   1008
1003        26 05          BNE  LOW     -1000
1005        C6 10          LDAB 10    -------
1007        4A             DECA            08
1008        20 F6          BRA  UP        +01
100A        39       LOW   RTS        -------
                                           09

                                           FF
                                          -09
                                      -------
                                           F6
```

The following is presented for those who prefer to let the computer, do the work. The catch is, if you are in memory, writing a program, it is bad to jump out and call a machine program, to do it for you. Therefore; HAVE IT BOTH WAYS.....

RELATIVE BRANCH CALCULATOR

By: Dale Heatherington

Calculating Relative Addresses

The calculation of relattve addresses for branch instructions on the Motorola 6800 (or Zi-log Z-80 or MOS 650X) may appear formidable at first to the beginner. However, this skill is required in order to manually code machine language instructions for short routines, as is often done to avoid the time required for an assembly, or in order to modify existing routines which have already been laodod into memory. The procedure for manually calculating short jumps (and most are short) is relatively simple, however. It involves a counting processes.

First consider jumps forward in the following routine:

```
0100   86  0200     LDAA    Load A with contents of 0200.
0103   81  45       CMPA    Is the value 45?
0105   27  -        BEQ     If equal go to 010A.
0107   20  -        BRA     if not, go to 0100.
0109   4C  -        INCA    Add 1 to A.
010A   BD  0400     JSR     Go to subroutine at 0400.
```

In this routine the relative address for BEQ (27 __) to jump forward to 010A is counted as follows:

```
        (00) (01)
0107     20   --
        (02)
0109     4C
        (03)
010A     BD   0400
```

Start counting at the next byte after the address with 00 and count forward to the desired address, in this case 03, so the correct instruction is 27 03.
So the correct address for the blank is F7. This will cause the jump back to 0100 as desired.

Remember: to jump forward, count forward from the relative address starting with 00 for the next byte, and to jump backward, start with the relative address byte as FF and count backward to the desired address. Double-check backward branches by counting forward from the calculated value, hopefully ending up with FF for the branch value. Double-check forward branches by counting forward again, as noted before.

If, in calculating a long branch, the resultant value cannot be contained in one byte (greated than FF for a forward branch or less than 00 for a backward branch), the branch is illegal as stated and must be changed to a jump or a chain of branches.

If a Motorola 6800 is available during the coding process, this same process may be automated by the use of the program below. The program starts at hex 0400 and is 112 bytes long. It makes use of two MIKBUG I/O subroutines, BADDR at E047 and PDATA1 at E07E.
When the program is started it will type "FROM". You should then type the address of the second byte of the branch instruction. Typing any character other than a valid hex digit will cause a return to MIKBUG. The program will then type "TO". You should then type in the target address of the branch. The program will then respond with the word "BRANCH" followed by the offset to be entered as the second byte of the branch instruction. If the target address is out of range the work "ERROR" will be printed. The program will loop back to its start and print "FROM" again. This process will be repeated until the RESET button is depressed

For backward branches, the counting is similar, except start

```
counting with the relative address byte as FF as follows:

              (F7)    (F8)  (F9)

     0100      B6      0200
              (FA)     (FB)

     0103      81       45
              (FC)     (FD)

     0105      27       03
              (FE)     (FF)

     0107      20       --           BRA      Go to 0100.

or a non-hex character is entered.
```

```
                              NAM     CAL
                       *RELATIVE BRANCH
                       *CALCULATOR FOR 6800
                       *BY DALE HEATHERINGTON
                       * 10/14/76
                       *MIKBUG ROUTINES AT $E047
                       * AND $E07E ARE USED.
                              OPT     O
                              OPT     S
0400                          ORG     $0400
      04           STK        EQU     $0498
0400  8E  0498     START      LDS     #STK          START PROGRAM HERE
0403  CE  0456                LDX     #FROM         PRINT "FROM"
0406  BD  E07E                JSR     $E07E
0409  BD  E047                JSR     $E047         INPUT ADDRESS
040C  FF  047B                STX     SAVE          STORE IT
040F  CE  045E                LDX     #TO           PRINT "TO"
0412  BD  E07E                JSR     $E07E
0415  BD  E047                JSR     $E047         INPUT ADDRESS
0418  FF  047D                STX     SAVE + 2      STORE IT
041B  8D  14                  BSR     COMPUTE       COMPUTE BRANCH
041D  CE  0466                LDX     #BRANCH       PRINT "BRANCH"
0420  BD  E07E                JSR     $E07E
0423  CE  0480                LDX     #SAVE + 5     PRINT VALUE OF BRANCH
0426  BD  E0BF                JSR     $E0BF
0429  CE  0470                LDX     #CRLF         OUTPUT CARRAGE RET
042C  BD  E07E                JSR     $E07E
042F  20  CF                  BRA     START         LOOP BACK TO THE START
0431  CE  047B     COMPUT     LDX     #SAVE         INDEX POINTS TO STORAGE AREA
0434  A6  01                  LDA A   1,X           GET "FROM" ADDRESS TO A&B
0436  E6  00                  LDA B   0,X
0438  43                      COM A                 INVERT ALL BITS IN A & B
0439  53                      COM B
043A  AB  03                  ADD A   3,X           DO A 16 BIT ADD WITH
043C  E9  02                  ADC B   2,X           THE "TO" ADDRESS
043E  A7  05                  STA A   5,X           STORE LSB OF RESULT
0440  C1  FF                  CMP B   #$FF          COMPARE MSB OF RESULT W/FF HEX
0442  27  07                  BEQ     BACK          BRANCH IF EQUAL
0444  5D                      TST B                 TEST B
0445  26  08                  BNE     ERR           BRANCH IF NOT ZERO
0447  4D                      TST A                 TEST LSB OF RESULT
0448  2B  05                  BMI     ERR           ERROR IF MINUS
044A  39                      RTS                   NO ERROR, RETURN
044B  4D           BACK       TST A                 TEST LSB OF RESULT
044C  2A  01                  BPL     ERR           ERROR IF PLUS
044E  39                      RTS                   NO ERROR, RETURN
044F  CE  0473     ERR        LDX     #ERROR        PRINT "ERROR"
0452  BD  E07E                JSR     $E07E
0455  39                      RTS                   RETURN
                       *MESSAGE AREA
0456  0D                 FROM FCB     $0D,$0A
0457  0A
0458  46                      FCC     5,FROM
0459  52
045A  4F
045B  4D
045C  20
045D  04                      FCB     4
045E  0D           TO         FCB     $0D,$0A
045F  0A
0460  20                      FCC     5,  TO
0461  20
0462  54
0463  4F
0464  20
0465  04                      FCB     $4
0466  0D           BRANCH     FCB     $0D,$0A
0467  0A
0468  42                      FCC     7,BRANCH
0469  52
046A  41
046B  4E
046C  43
046D  48
046E  20
046F  04                      FCB     4
0470  0D           CRLF       FCB     $0D,$0A
0471  0A
0472  04                      FCB     4
0473  0D           ERROR      FCB     $0D,$0A
0474  0A
0475  45                      FCC     5,ERROR
0476  52
0477  52
0478  4F
0479  52
047A  04                      FCB     4
047B  00           SAVE       FCB     0
                              END
```

MAILLIST

Randal Lilly N3ET
752 S. Carldon St.
Allentown, PA 18103

OK so you can't get away from the computer games long enough to make use of the $3,000.00 toy. Well you can help the local radio club or other society to keep those books straight. This MAILLIST program was written to keep track of a newsletter mailing list. As set up here, it will produce mailing labels and a formatted list which is used when updating is needed.

While the program as is does a fine job, it can be readily changed for other output formats. The label formatter prints four lines of address and two blanks lines for each label. A strip of blank labels and a Decwriter are used for printing. The number of printable lines may be changed at location $0086 and the number of blank lines may be changed by adding more or less JSR PCRLF statements at location $0135.

The listing formatter prints similar to that of the label formatter, except that it prints three labels across, ten deep, does paging and has a private file option. The number of lines printed and the width should not be changed. The depth, which is set to ten, may be changed at location $00CE, and the number of margin (eject) lines may be changed at location $00C6. Location $00F2 may be used to make minor changes in the field width of each address.

Use of the TSC Text Editor (C) is invaluable when inputting a new file or updating an existing file. The address files are stored in FLEX (TM) minidisk files, and are compatible with the text editor, FLEX BUILD utility or SWTPC BASIC.

Using the TSC Text Editor, a typical edit of the address file will look like the following:

```
 1.00 LASTNAME, FIRSTNAME
 2.00 STREET ADDRESS
 3.00 CITY, STATE
 4.00 ZIP    HAM CALL
 5.00 CODES  *PHONE
 6.00
 7.00 LASTNAME, FIRSTNAME
 8.00 STREET ADDRESS
 9.00 CITY, STATE
10.00 ZIP    HAM CALL
11.00 CODES  *PHONE
12.00
```

The last entry must be followed by

twelve blank lines and a line containing one or more 'UP ARROWS'. The 'UP ARROW' character tells the MAILLIST program that it should finish. The twelve blank lines look like two blank entries. These are printed as blanks.

Line 11 above can be used for special codes as you see fit and these will be printed. Line 11 also shows how an unlisted (private) phone number is treated. When a '*' character is found, it and that which follow it are not printed, unless you indicate it is not a PRIVATE file, when asked at the start of the program. Lines 6 and 12 are normally blank, but may contain other information (which will not be printed). Only the formatted list will print lines 5 and 11, so the mailing labels will not have phone numbers on them.

To use the program type MAILLIST, FILENAME. The text file will be pulled off disk and stored in memory. The program will ask if it is a private file, type 'Y' if you do not want anything following the '*' to be printed. Otherwise type 'N'. Next, type 'M' followed by a carriage return for mailing labels, or 'L' followed by a carraige return for a formatted list.

Try a small file first (about six names and addresses) and run the program. Then get busy and enter all the names and addresses you might have. Later updates become easier each time.

Editors Note: The 'formatted list' can also be labels (Avery #5380) by setting the field width of each line (Line 185, $00F2).

```
PAGE  001  MAILLIST    R. LILLY 1-4-79

00001                      NAM    MAILLIST R. LILLY 1-4-79
00002               *R.LILLY 1-4-79
00003               *MOVE TEXT TO MEMORY,FORMAT,PRINT
00004 0000                 ORG    $0000
00005      6FFF     MEMEND EQU    $6FFF
00006      7103     WARMS  EQU    $7103
00007      E1AC     INEEE  EQU    $E1AC
00008      7112     PUTCHR EQU    $7112
00009      711E     PCRLF  EQU    $711E
00010      7127     GETFIL EQU    $7127
00011      712D     SETEXT EQU    $712D
00012      7740     FCB1   EQU    $7740
00013      7806     FMS    EQU    $7806
00014      7803     FMSCLS EQU    $7803
00015      E07E     PDATA1 EQU    $E07E
00016 0000 20 05           BRA    TOM
00017 0002 01       VN     FCB    1          VERSION NUMBER 1
00018 0003          XSAVE  RMB    2          STORE ADDRESS
00019 0005          LINCNT RMB    1
00020 0006          COUNT  RMB    1
00021               *
00022               *LINES OF TEXT MUST BE
00023               *FOLLOWED BY C.R.'S AS IN A TEXT
00024               *EDITOR DISC FILE.
00025               *
00026               *TYPE 'MAILLIST.FILENAME'
00027               *FILE PULLED FROM OPPOSITE DRIVE,
00028               *UNLESS DRIVE IS SPECIFIED.
00029               *
00030               *TYPE M C.R. (MAILING LABELS)
00031               *TYPE L C.R. (FORMATTED LIST)
00032               *
00033               *IF A PRIVATE FILE, THE TEXT FOLLOWED
00034               *BY A '*' WILL NOT BE PRINTED.
00035               *
00036               *TEXT SHOULD BE AS FOLLOWS:
00037               *  NAME        LILLY, RANDY
00038               *  STREET      752 S. CARLOON ST.
00039               *  TOWN        ALLENTOWN, PA.
00040               *  ZIP         18103   N3ET
00041               *  INFO        TCK 8791-3774 (PRIVATE)
00042               *---- BLANK LINE -----------
00043               *
00044               *END OF FILE SHOULD CONTAIN
00045               *12 BLANK LINES FOLLOWED BY A LINE
00046               *OF '^^^^^^^^^^^' (UP ARROWS)
00047               *
00048               *LABELS ARE 4 LINES DEEP WITH
00049               *2 BLANK LINES.
00050               *
00051               *FORMATTED LIST IS 3 WIDE BY
00052               *10 DEEP AND DOES PAGING.
00053               *
00055               *
00056               * TEXT TO MEMORY
00057               *
00058 0007 CE 0155 TOM    LDX    #BUFST
00059 000A DF 03          STX    XSAVE
00060 000C CE 7740        LDX    #FCB1
00061 000F BD 7127        JSR    GETFIL   GET FILE SPEC
00062 0012 25 14          BCS    ERROR
00063 0014 86 01          LDA A  #1       SETUP TXT EXT
00064 0016 CE 7740        LDX    #FCB1
00065 0019 BD 712D        JSR    SETEXT   SET DEFAULT EXT
00066 001C CE 7740        LDX    #FCB1
00067 001F 86 01          LDA A  #1       OPEN FOR READ COMMAND
00068 0021 A7 00          STA A  0,X
00069 0023 BD 7806        JSR    FMS
00070 0026 27 1F          BEQ    NEXTCH
00071              *ERROR
00072 0028 CE 011F ERROR  LDX    #ERRORO
00073 002B BD E07E        JSR    PDATA1
00074 002E 7E 7103 OUT    JMP    WARMS
00075              *
00076 0031 BD E07E OUTIN  JSR    PDATA1   CRT OUTPUT
00077 0034 7E E1AC INE    JMP    INEEE    CRT INPUT
00078              *
00079 0037 DE 03   OK     LDX    XSAVE
00080 0039 A7 00          STA A  0,X      STORE BYTE IN MEMORY
00081 003B A1 00          CMP A  0,X      TEST MEMORY
00082 003D 26 16          BNE    CLOSE
00083 003F 08             INX
00084 0040 DF 03          STX    XSAVE
00085 0042 8C 6FFF        CPX    #MEMEND
00086 0045 27 0E          BEQ    CLOSE
00087 0047 CE 7740 NEXTCH LDX    #FCB1
00088 004A BD 7806        JSR    FMS      GET NEXT RECORD
00089 004D 27 E8          BEQ    OK
00090 004F A6 01          LDA A  1,X      GET BYTE FROM FLEX
00091 0051 81 08          CMP A  #8       END OF FILE?
00092 0053 26 D3          BNE    ERROR
00093 0055 BD 7803 CLOSE  JSR    FMSCLS   CLOSE FLEX FILE
00095 0058 8E A042 MAIL   LDS    #$A042
00096 005B CE 012C        LDX    #PRIVAT  PRIVATE FILE?
00097 005E 8D D1          BSR    OUTIN
00098 0060 5F             CLR B
00099 0061 81 59          CMP A  #'Y
00100 0063 26 02          BNE    STAB
00101 0065 C6 2A          LDA B  #'*
00102 0067 D7 02   STAB   STA B  VN
00103 0069 CE 0144 PROMP  LDX    #LISTIT  LABEL OR LIST?
00104 006C 8D C3          BSR    OUTIN
00105 006E 16             TAB
00106 006F 8D C3          BSR    INE
00107 0071 81 0D          CMP A  #$0D
00108 0073 26 F4          BNE    PROMP    NOT CR
00109 0075 17             TBA
00110 0076 81 1B          CMP A  #$1B     ESCAPE
00111 0078 27 B4          BEQ    OUT
00112 007A 81 4C          CMP A  #'L
00113 007C 27 2B          BEQ    LIST     PRINT FORMATTED LIST
00114 007E 81 4D          CMP A  #'M
00115 0080 26 D6          BNE    MAIL     IMPROPER ENTRY
00116              *
00117              * GENERATE MAILLING LABELS
00118              *
00119 0082 CE 0154 LABELS LDX    #BUFST-1
00120 0085 C6 04   MAIL2  LDA B  #4       PRINT 4 LINES
00121 0087 08      MAIL1  INX
00122 0088 A6 00          LDA A  0,X
00123 008A 81 5E          CMP A  #'^      END OF PRINTING
00124 008C 27 6F          BEQ    EXIT
00125 008E 81 0D          CMP A  #$0D     C.R. ?
00126 0090 27 05          BEQ    NLINE
00127 0092 BD 7112        JSR    PUTCHR
00128 0095 20 F0          BRA    MAIL1
00129              *
00130 0097 BD 711E NLINE  JSR    PCRLF    C.R.-L.F.
00131 009A 5A             DEC B
00132 009B 26 EA          BNE    MAIL1
00133 009D C6 02          LDA B  #2       SKIP 2 LINES OF TEXT
00134 009F 8D 4A          BSR    FALL
00135 00A1 BD 711E        JSR    PCRLF    BLANK LINES
00136 00A4 BD 711E        JSR    PCRLF    BETWEEN LABELS
00137 00A7 20 DC          BRA    MAIL2
00139              *
00140              * GENERATE FORMATTED LIST
00141              *
00142 00A9 BD 711E LIST   JSR    PCRLF
00143 00AC CE 0154        LDX    #BUFST-1
00144 00AF C6 05          LDA B  #5       5 LINES TO BE PRINTED
00145 00B1 D7 06          STA B  COUNT
00146 00B3 20 1B          BRA    SETPAG   INIT LINCNT
00147              *
00148 00B5 C6 05   GROUP  LDA B  #5       5 LINES TO BE PRINTED
00149 00B7 D7 06          STA B  COUNT
```

```
0089 C6 0D           LDA B  #13       SKIP 13 LINES OF TEXT
008B 8D 2E           BSR    FALL
008D BD 711E         JSR    PCRLF     SKIP LINE
00C0 7A 0005         DEC    LINCNT
00C3 26 0C           BNE    PRINT3    NOT TIME FOR MARGIN
  C5 C6 06           LDA B  #6        6 MARGIN LINES
00C7 BD 711E MARGIN  JSR    PCRLF     OUTPUT C.R.-L.F.
00CA 5A              DEC B
00CB 26 FA           BNE    MARGIN
00CD C6 0A   SETPAG  LDA B  #10       PRINT 10 DEEP
00CF D7 05           STA B  LINCNT    INZ LINE COUNT
00D1 8D 1E   PRINT3  BSR    PRINT     PRINT PARTIAL LINE
00D3 DF 03           STX    XSAVE
00D5 8D 12           BSR    FALL5     SKIP NEXT 5 LINES OF TEXT
00D7 8D 18           BSR    PRINT     PRINT PARTIAL LINE
00D9 8D 0E           BSR    FALL5     SKIP NEXT 5 LINES OF TEXT
00DB 8D 14           BSR    PRINT     FINISH PRINTING LINE
00DD BD 711E         JSR    PCRLF     END OF LINE
00E0 DE 03           LDX    XSAVE
00E2 7A 0006         DEC    COUNT
00E5 27 CE           BEQ    GROUP
00E7 20 E8           BRA    PRINT3
             *
             * FALL THROUGH TEXT
             *
00E9 C6 05   FALL5   LDA B  #5        SKIP 5 LINES OF TEXT
00EB 8D 2A   FALL    BSR    FINDCR    SEARCH FOR C.R.
00ED 5A              DEC B
00EE 26 FB           BNE    FALL
00F0 39              RTS
             *
             * PRINT PARTIAL LINE
             *
             *
00F1 C6 1B   PRINT   LDA B  #27       26 CHARACTERS (CRT=HEX 17)
00F3 08              INX
00F4 5A      PRINTA  DEC B
00F5 27 1F           BEQ    FIND1     SEARCH FOR C.R.
00F7 A6 00           LDA A  0,X
00F9 81 5E           CMP A  #'^       LOOK FOR END
00FB 26 08           BNE    CKPR
00FD 86 07   EXIT    LDA A  #7        BELL
00FF BD 7112         JSR    PUTCHR
0102 7E 0058         JMP    MAIL
0105 91 02   CKPR    CMP A  VN        LOOK FOR  O-P INT CHARACTER
0107 27 04           BEQ    OUTSPC
0109 81 0D           CMP A  #$0D
010B 26 03           BNE    SKSPC
010D 86 20   OUTSPC  LDA A  #$20
010F 09              DEX
0110 08      SKSPC   INX
0111 BD 7112         JSR    PUTCHR
0114 20 DE           BRA    PRINTA
             *
             * SEARCH FOR NEXT C.R.
             *
             *
0116 09      FIND1   DEX
0117 86 0D   FINDCR  LDA A  #$0D
0119 08      FIND    INX
011A A1 00           CMP A  0,X
011C 26 FB           BNE    FIND
011E 39              RTS
             *
011F 4E      ERRORO  FCC    'NO SUCH FILE'
012B 04              FCB    4
012C 49      PRIVAT  FCC    'IS THIS A PRIVATE FILE?'
0143 04              FCB    4
0144 0D0A    LISTIT  FDB    $D0A
0146 4D              FCC    'Mail or List ?'
0154 04              FCB    4
     0155    BUFST   EQU    *
0155 52              FCC    'R. LILLY 1-4-79'
                     END              TOM
TOTAL ERRORS 00000

SYMBOL TABLE
BUFST  0155  CKPR   0105  CLOSE  0055  COUNT  0006  ERROR  0028
ERRORO 011F  EXIT   00FD  FALL   00EB  FALL5  00E9  FCB1   7740
FIND   0119  FIND1  0116  FINDCR 0117  FMS    7806  FMSCLS 7803
GETFIL 7127  GROUP  00B5  INE    0034  INEEE  E1AC  LABELS 0082
LINCNT 0005  LIST   00A9  LISTIT 0144  MAIL   0058  MAIL1  0087
MAIL2  0085  MARGIN 00C7  MEMEND 6FFF  NEXTCH 0047  NLINE  0097
OK     0037  OUT    002E  OUTIN  0031  OUTSPC 010D  PCRLF  711E
PDATA1 E07E  PRINT  00F1  PRINT3 00D1  PRINTA 00F4  PRIVAT 012C
PROMP  0069  PUTCHR 7112  SETEXT 712D  SETPAG 00CD  SKSPC  0110
SYMBL  0067  TOM    0007  VN     0002  WARMS  7103  XSAVE  0003
```

MODEMS

Reprint from '6800 BITS'
Chicago Area 6800 Newsletter
Phil Schuman, Editor

Several questions have come up since we ran the story about the 'Kansas City Phone Patch'. People wanted to know, why they could not access the CBBS? To answer this, and probably several other questions about modems, let's start at the beginning.

What is a modem? This is a contraction for the name 'modulator/demodulator'. What are these animals? These animals were created to transmit data over common carrier circuits such as phone lines and microwave. It is easy to transmit data over wires, but what do you do if it is to be sent over radio, such as microwave? The answer is to convert the voltage pulses into pulses of tones, this is what the modem does.

A set of standard tone frequencies and protocol have been established by the Bell System and everyone else (for the most part) remains compatible. In addition to these tones, the connections to the modem are standard also; RS-232. Now the question is how to hook up the terminal you have to the computer at the other end of the world. A modem is placed at each point: a good start. Between the modems we place a microwave link. Now the question, how do we connect to the modems at each end? Well that depends, who is calling who. For the most part, the terminal (and you) will be placing the 'call' to the computer. And thus are referred to as the 'originator'. The computer at the other end will therefore be called the 'answer'.

Next question. How many wires connect to your terminal using the RS-232 protocol? Well, there is; transmit data and it's ground, receive data and it's ground. In this case the grounds are really only one, but logically both are needed. We therefore logically need 4 wires; hence we will need 4 tones. Logical high and logical low for both directions; to and from the terminal. This is called 'full-duplex' because data can travel in both directions at the same time.

As far as the tones are concerned, for the 'originate mode' they are:

Receive:

1270 mark/logic high
1070 space/logic low

Transmit:

2225 mark/logic high
2025 space/logic low

For the modem at the other end operating in 'answer' mode, the tone frequencies are mirror images of the ones above.

In addition, for those using a phone line, the answer modem will send an 'answerback' tone to signal that the connection has been established. This tone is 2025 hz, and will also disable the echo suppressors on the phone circuit. The phone lines only send voices one way at a time...in case you did not know that, and the tone will disable that feature. O.K. what is this animal called? It is the Bell System 103 modem configuration.

This can only transmit data up to about 600 baud, before the bandwidth is not capable of switching at the proper baud rate.

What can be done now, is to drop one set of tones, and use the phone line one-way at a time; half-duplex. This form of transmission can handle up to 1200 baud, but more intelligence is needed to control the modems. The tone frequencies for the Bell System 202 modem configuration are:
1200 and 2200 with 2025 for answerback.

The KCPP uses the SWTPC AC-30 for it's tone generation, and the tones for the AC-30 are:
1200 and 2400
They are only half-duplex and do not fit in with any of the data transmissions standards.

I hope that this has proven informative to those that were wondering about modems and how they work or are used. Most of the systems that 'talk', use the Bell 103 standard at 300 baud or 30 cps.

RS-232 Connectors

For those who would like to change over to the more common RS-232 connectors, here is the pin configuration and their uses. Logic levels (high/low) are represented by either +12v or -12v. Actually the voltages may be in the range of 3 to 25v. Most signals are represented by being ON with a positive (+) voltage and OFF with a minus (-) voltage.

Each pin has a two character EIA code, along with a common mnemonic. Most of the pins will be listed first, and then a condensed version of those that are actually needed. Please keep in mind, that this interface was designed to connect a terminal to a modem.

1. AA FG Chassis Ground
2. BA TD Trans Data from Terminal
3. BB RD Rec Data from Computer
4. CA RTS Request to send
5. CB CTS Clear to Send
6. CC DSR Data Set Ready
7. AB SG Signal Ground
8. CF DCD Data Carrier Detect
   Modem tones receive
9. +12v for testing
10. -12v for testing
15. DB TC Transmit Clock
17. DD RC Receive Clock
20. CD DTR Data Terminal Ready
   Terminal is 'on-line'
22. CE RI Ring Indicate
   Phone is ringing
24. DA ETC External Trans Clock
25 CN Busy
   Phone is dialing out

Here are the pins that are needed for most terminal configurations.
2. Trans data to Modem/CPU
3. Receive date from CPU
7. Ground

In addition the following pins may have to be jumpered, to force the terminal into a 'Ready'; 5, 6, 8, 20. Clocking may be tied to the following:
17. Clock from CPU into terminal
24. Clock from terminal into CPU

NEW PRODUCTS

TSC BASIC for the 6800

Technical Systems Consultants, Inc. is pleased to announce the availability of the TSC BASIC for the 6800. The program resides in 9.5K of memory and is currently the fastest floating point BASIC interpreter available for any 8 bit micro. Typical speed increases of 2 to 10 times have been observed in comparison to other popular BASIC's, with some cases up to 75 times! All of the standard BASIC statements and functions are supported as well as many extended capabilities. Both floating point and string variables are provided with strings being fully dynamic and unrestricted in size. Variable names may be either the standard types or double letter combinations allowing limited variable name mnemonics.

Other features include single and double dimensioned arrays. Array references support subscripts of 0 unlike several other 6800 BASIC's available. Array size, loop nesting, subroutine nesting and string length are only limited by the amount of user memory available in the machine. A tremendous enhancement is provided by the 'IF..THEN..ELSE' construct. The 'ELSE' clause promotes a more structured type programming style, thus improving readability and conciseness of the program. The input buffer allows lines as long as 127 characters to be entered to take advantage of the complex statement structures permitted with this statement. Other features include the HEX function which allows hexadecimal number representation while PI provides an easy reference to this often used constant. The floating point arithmetic done by BASIC is performed to seven digits accuracy internally, with all answers printed to six. The dynamic range of the numbers is in the range of 10 raised to the plus or minus 37th power.

Overall, TSC BASIC is a very fast and powerful BASIC. It is easily adapted to run in any 6800 system having at least 12K of user RAM available from location 0000. A system with 16K or more of memory is recommended for serious applications. The BASIC is available on Kansas City Standard cassette along with a complete user's manual for $39.95. No source listing is available. A full disk file version of TSC BASIC which will run under the FLEX™ disk operating system will be available shortly. The product is available from stock and may be ordered from Technical Systems Consultants, Inc., Box 2574, West Lafayette, Indiana, 47906, or call (317) 463-2502.

I. General

Memory required: 9.5K, 16K recommended
Arithmetic precision: 6 digits, 7 internally
Dynamic range: approx. 10E+37 to 10E-38
String storage: Fully dynamic, no limit to size
Array storage: 1 and 2 dimensional, no size limit
Nesting: No limit to GOSUB & FOR nests
Multiple statements per line: With ':' or '\'
Input line buffer: 127 characters
Variable types: floating point and String
Variable names: Single letter, 2 letters, or letter+number

II. Commands

CLEAR	CONT	EXIT	LIST
LOAD	NEW	RUN	SAVE

III. Statements

DATA	DEF	DIM	END	FOR	GOSUB
GOTO	IF..THEN	IF..GOTO	IF..ELSE	INPUT LINE	INPUT
LET	NEXT	ON..GOSUB	ON..GOTO	POKE	PRINT
READ	REMARK	RESTORE	RETURN	STEP	STOP

IV. Arithmetic Functions

ABS	ATN	COS	EXP	FRE	INT
LOG	PEEK	POS	RND	SGN	SIN
SPC	SQR	TAB	TAN	USR	PI

V. String Functions

ASC	CHR$	HEX	LEFT$	LEN	MID$
RIGHT$	STR$	VAL			

SEMICONDUCTOR
Memory Primer

Don Kinzer
3885 NW Columbia Ave.
Portland, OR 97229

As one might guess from the preceeding paragraph PC board layout is extremely critical. Indeed, a wirewrap prototype is almost useless because of transients. Another critical area is power supply bypassing. The dynamic RAMS draw almost no quiescent (DC) current. Power is consumed at and shortly after the transistions of the three clocks RAS, CAS, and WRITE. The transient currents on the 3 power supply lines and ground may exceed 100 mA on each chip for 50 to 100 ns duration. These current demands must be met by the local bypassing capacitors in order to reduce $L\frac{di}{dt}$ voltage drop (noise) on the relatively long supply traces and their associated parasitic inductance.

The 4116 memory requires that each of the 128 internal rows (represented by the seven address bits at RAS) be refreshed at least every 2 ms. However, since the chips consume power only when cycling it is desirable to keep refreshes to a minimum to conserve power.

Since the 6800 does not access memory during $\emptyset_1$ it is very natural to perform refreshes at that time. Using this technique the refresh is said to be transparent as it does not affect processor speed by delaying accesses. The refresh is also synchronous because it is synchronized to the processor.

Referring again to the schematic of Figure 8, the oscillator composed of R_1, R_2, R_3, Q_1, C_1, and U7E clocks the counter U10 at about 1 MHz. This causes the carry out to go high after 15 cycles (assuming 0 initially) or about 15 microseconds. 128

refresh cycles will occur in just under 2 ms worst case, meeting the refresh requirements while keeping the refreshes to a minimum.

Refresh pending is sampled on every $\overline{\emptyset_2}$ ($\emptyset_1$) by U6A in order to guarantee set up and hold times on the REFRESH flip flop U6B (thereby avoiding glitches) which is clocked by $\emptyset_1$ ($\overline{\emptyset_2}$). As soon as REFRESH goes true the refresh time counter is cleared to begin timing the next 15 microsecond cycle. Concurrently, a pulse is started down the delay line by U6B via U8B and U9A,B. Also, the REF ENABLE line is asserted on the 3242 (U3) causing the current contents of the refresh address register (7 bits) to be multiplexed onto it's A_{6-0} outputs which are applied directly to the RAMs.

When the pulse gets to the 40 ns tap of the delay line U8C will cause RAS to be asserted on every RAM chip. In this manner every chip is refreshed simultaneously at a given row address.

As the pulse reaches the 200 ns tap the REFRESH flip flop is reset via U7G. This action de-asserts RAS and also increments the refresh addrss internal to the 3242 via the COUNT line. Simultaneously, a rising edge is injected into the delay line to to return it to its idle state which completes the refresh cycle.

There are many seemingly minor details that if heeded will produce a good, reliable dynamic memory system. Ignoring the details will generate more headaches than the designer bargained for.

The first, and perhaps most important, consideration is power distribution which was briefly touched upon earlier. The V_{DD}

(+12) supply sources most of the current required by the 4116. Recalling the need for distributed low inductance capacitance, the most logical procedure is to opt for a multi-layer circuit board. The center two layers would be V_{DD} and V_{SS} (ground). This provides both a good ground plane and a good low resistance path for the V_{DD} supply. The two parallel layers also provide a surprising amount of distributed capacitance. Every 4116 should also have its own high quality low ESR, low inductance capacitor (such as ceramic) in the .01 µF to .1 µF range. These should be located as close as physically possible to the RAM chip to reduce the parasitic inductance.

The other two supplies V_{CC} (+5) and V_{BB} (-5) should be run in a matrix fashion on the outer board layers using the widest possible runs. Each chip should have its own bypass capacitor on each of these supplies also, again as close as possible to the chip. Additionally, all of the supplies should have several bulk bypass capacitors of 5 to 10 µF distributed about the board. Again the order of importance of the supplies is $V_{DD} > V_{BB} > V_{CC}$ and the bulk capacitance should vary accordingly.

The next area of concern is the logic inputs to each memory chip (RAS, CAS, WRITE, A_{6-0}, D_{in}). These signals must be kept from ringing as much as possible. Overshoot can cause all kinds of problems ranging from data loss to chip destruction. The way to keep ringing to a minimum is to keep parasitic inductance low by keeping runs as short and direct as possible. Furthermore, adding series damping resistors close to the drivers in the 10 ohm to 60 ohm range will help. The best value is determined by experimentation.

The clock signals (RAS, CAS, and WRITE) must be generated glitch-free and the asynchronous inputs (A_{6-0}, D_{in}) must not be allowed to change within the set up and hold times specified. To minimize cross talk between RAS and CAS they should be run at right angles to one another, i.e. one on the front layer, one on the back.

An error which is commonly found in memory systems is that the designer used the system reset signal from the bus to reset the synchronous elements (flip flops, counters, etc.) in the controller circuit. While this appears to be the logical thing to do it allows a front panel reset, which is asynchronous with respect to the controller, to abort an access or refresh cycle which will almost always cause data loss. Once a cycle is begun it must be taken to completion.

This requirement constrains the designer to either use an on-board power-on-reset or to design the sequential logic so that it is self starting from power up. Furthermore, the bus signals which initiate a cycle must be latched so that their premature removal will not abnormally terminate the cycle.

The bias supply V_{BB} is used to bias the substrate of the RAM chips. This voltage increases the breakdown potential of the MOS transistors and has other benefits as well. Since at power up the V_{DD} and V_{CC} supplies may overshoot steps must be taken to limit overshoot (i.e. clamping) and the supplies should be sequenced on and off such that V_{BB} is applied first and removed last with respect to V_{DD} and V_{CC}.

Many memory systems include circuitry for data error detection using parity. To implement this an extra RAM chip is added to

each row to retain the parity bits. As each word is written to, the parity of the data is generated and the result written to a corresponding address in the parity RAM. On a read the parity of the read data is again generated and compared to the contents of the parity RAM at that location. If they don't match an interrupt is generated to the CPU. It is also possible to implement error correction in a memory system such that errors of a certain type are automatically corrected on the fly.

In summary, we have seen that there are several features which characterize any type of storage medium. Furthermore, we have through a typical design example seen how to implement a memory system using the most misunderstood and most feared memory element, the dynamic RAM.

References

MK 4116 Data Sheet, "Mostek 1977 Memory Products Catalog", Mostek Corporation, pp. IV-66 to IV-75.

3242 Data Sheet, "Intel Data Catalog 1977", Intel Corporation, pp. 5-29 to 5-32.

"Intel Memory Design Handbook", Intel Corporation, 1975.

Frankenberg, R.J., "Designer's Guide to: Semiconductor Memories"-
Part I, EDN, August 5, 1975, pp. 22-29
Part II, EDN, August 20, 1975, pp. 58-65
Part III, EDN, September 5, 1975, pp. 68-74
Part IV, EDN, September 20, 1975, pp. 62-67.

Calebotta S., "System Design with Dynamic Memory Takes Attention To Detail", Electronics, February 2, 1978, pp. 109-113.

Mostek Corporation, "Semiconductor Memory Design", Digital Design, April 1978, pp. 40-50.

THE TERMINAL-

Until recently all terminal functions were designed with hardware logic. A relatively simple terminal with limited functions could easily require as many as sixty or more integrated circuits. More sophisticated terminals with a moderate amount of intelligence could easily have over a hundred IC's. All this has now changed. With the introduction of MOS video controller circuits it has become possible to design a terminal using a controller and a microprocessor that will perform almost any imaginable function with software. The CT-82 has one hundred twenty-eight separate functions—all of which are software driven. It contains fewer parts than most "dumb" terminals.

The normal screen format is 16 lines (20 lines selectable) with 82 characters per line. This is an upper-lower case display with a 7 x 12 dot matrix. The high resolution characters are displayed on a Motorola Data Products M-2000 series monitor with a green P-31 phosphor. This monitor has a 12 MHz video bandwidth and dynamic focus circuits to insure a crisp well focused display over the entire face of the tube. An alternate all capital letter format is available (optional) with 16, 20 or 22 lines and 92 characters per line. The lower case portion of this character set has graphic symbols. In this mode the lines may be moved together to give a solid figure or line. Direct cursor addressing combined with the plotting capability makes it possible to indicate the end points of a line and then to automatically draw a line between them.

Both the monitor and the character generator have sockets provided for alternate material in the form of an EPROM. This makes it possible to have special terminal functions, or character sets that can be switched in under computer control.

The CT-82 has its own internal editing functions. This allows inserting and deleting lines and characters, erasing quadrants, or lines; doing rolls, scrolls, slides and other similar functions. The CT-82 can block transmit completed material to the computer, or output material to its own remote printer through the built-in parallel printer I/O port. The terminal can be programmed to operate at any system baud rate that is normally used from 50 to 38,400. The baud rate may be changed at any time within this range with a software command.

The cursor position, type of cursor, cursor ON-OFF and blinking are all provided. A command is provided to print control characters and also to turn on and off a tape punch, or tape reader. Protected fields, shift inversion, dual intensity and many other miscellaneous features make the CT-82 one of the most flexible terminals available.

A fifty-six key alphanumeric keyboard plus a twelve key cursor pad is standard. A numeric pad may be substituted for the cursor pad (optional). Connection to the terminal is through a standard DB-25 connector and RS-232 signal levels. The CT-82 operates from 100, 115, 220, or 240 VAC at 50 to 60 Hz. It weighs 20 lbs, and is a compact 18" wide, 10" high and 18" deep.

CT-82 Intelligent Terminal
assembled and tested . . . $795.00 F.O.B. San Antonio

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. Rhapsody
San Antonio, Texas 78216
(512) 344-0241

THE EDITOR-

The only microprocessor editor with all the features and ease of use normally found only on large machines. "THE EDITOR" lets you fully use the CT-82's capabilities.

LINE POINTER —Now you understand why the CT-82 has 82 columns. The left two columns are used for a line pointer, which indicates the line of text being edited.

FILE WRAPAROUND—"THE EDITOR" may make multiple passes over the file being edited without restarting the editor.

AUTOMATIC CARRIAGE RETURN—The last word in a line will automatically be started on the next line if it will not fit in the space remaining on the line.

SIMPLE COMMANDS—Commands consists of a single letter, or a key press on the cursor pad. No complicated format to be learned and remembered.

MULTIPLE COMMANDS and REPEATS—Command line may have more than one command. "THE EDITOR" will execute command strings sequentially. Repeat function allows changes in a string through the text file.

SOURCE TEXT TABS—Tab stops appropriate for source text input may be set to operate from the space bar, or any other key.

SHIFT INVERSION—The keyboard may be set to produce either capital, or lower case letters when shift is used.

SCREEN POSITIONING—Scroll up, scroll down, line pointer up, line pointer down, home file, top of memory, bottom of memory, move relative to pointed line and form feed are provided.

"THE EDITOR" is available only for Southwest Technical Products computer systems using the CT-82 and running under FLEX-5®, or FLEX-8® operating systems. It may be used to edit any files, or programs compatible with the DOS, except binary files. Edited files are compatible with the TSC Text Processing program. The combination makes a powerful and inexpensive word processing system.

Editor FLEX-5® or FLEX-8® . . . . $25.00 ppd. in Continental USA

®FLEX is a registered trademark of TSC Inc.

SOUTHWEST TECHNICAL PRODUCTS CORP. 1964 SWTPC 1979 OUR 15TH YEAR

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. Rhapsody
San Antonio, Texas 78216 **(512) 344-0241**

LOCATE

A Utility Command Program for FLEX™

Robert L. Pigford
300 Wilson Rd.
Newark, Del. 19711

Users of the SWTPC 6800 disk operating system, "Flex", written by Technical Systems Consultants (TSC) of W. Lafayette, Ind., are accustomed to the versatility and convenience of this fine software. It makes the use of the computer much more satisfying than operation with tape. The Flex system incorporates on its master disk a large number of "utility command" routines for management of the disk files which the user may have stored on his disks. Among these subroutines is the disk command, SAVE, which permits the user to transfer to disk from memory any binary program which he may have developed in RAM. Such a program is readily and quickly moved from disk back into RAM by entering its file name on the user's terminal. However, the Flex system provides no way to fine either the locations in RAM at which the previously stored program begins and ends or its starting address for execution. After recording many programs the user has very likely forgotten what memory locations he used. An ability to find this information from the disk file itself is often very convenient.

LOCATE is such a program. Stored on the master disk with other command routines, it can be called at any time to display on the terminal the address in RAM at which the first and the last bytes of a program will be located and the address at which execution will begin. For example, if one wants to know the RAM addresses associated with a binary file named "Test" located somewhere on disk number one he simply types "LOCATE,1.TEST" and obtains the following output on his terminal:

```
LOCATE,1.TEST

PGM. STORAGE BEGINS AT 0100
PGM. STORAGE ENDS AT   1234
STARTING ADDRESS  =    0200
```

The default file extension assumed in LOCATE is .BIN. Other types of binary files, such as command and system files, can be located by adding the appropriate extension after the file name in the call for the LOCATE program. BASIC and ASCII text files can not be located because they do not include memory address data.

Table 1 is a listing of the program, as obtained from the TSC assembler. Extensive use is made of the "Get" and the binary "File Loader" routines already contained in the Flex file management system, each having been modified to count successive bytes in the stored program

rather than to transfer bytes to RAM. The "Locate.Cmd" program operates from a block of memory beginning at $7600 within the Flex master program, where most of the already supplied command routines also operate when they are called.

To understand the program itself one needs to know the format of binary files on disk. This is explained in the "Advanced Programmer's Guide", written by TSC and available from SWTPC. A binary record begins with a first byte containing a start-of-record indicator, 02. This is followed by the most and the least significant bytes of the load address in RAM at which the program will start. The next byte gives the number of data bytes which will follow in the record, after which the program data bytes themselves occur. On the last record occupied by the program the first byte is $16, indicating that what follows is the transfer address at which program execution will begin. The ending address to be used in RAM is not recorded in the disk file but is computed by LOCATE as it reads through the file, incrementally adding the total number of bytes in the program to the initial memory address.

Bytes are read from the disk by the file-management subroutine here called "FMS1". It returns in the A-register one byte of the file and an error signal, if one occurs. One such signal is an end-of-file marker byte, 08, which signals the subroutine that all information recorded in the file has been read. When this occurs, FMS1 returns control to the main program which begins at START, not to the LOADER subroutine. This is done by using two PUL A instructions simply to change the stack pointer so that a RTS instruction will return control to the previous program's subroutine jump, not to the subroutine which just previously caused entry into FMS1. This accounts for the absence of a RTS instruction at the end of LOADER.

The reader can easily assemble the "Locate" program using the listing given in the table. If he asks for a binary object file to be put on disk drive number one, for example, he may next use Flex to change the name of the new "locate" binary object file to 1.LOCATE.CMD . Then he can copy this file from drive number one to number zero if drive zero contains his collection of Flex user command programs. Now he should find it easy to obtain memory location information directly from the disk on which his programs are stored.

SYMBOL TABLE

ERROR	71BD	FLAG	7603	FMS1	7525	INIT	7609	INIT1	7606
JUMP	761F	LDR1	7625	LDR2	765D	LDR3	7689	LDR4	767E
LDR5	7672	LOADER	7622	MSG1	7698	MSG2	76AF	MSG3	76C6
OPEN	758A	OUT4HS	E0C8	PCRLF	711E	PSTRIN	7118	RETURN	7617
SCRATC	70B3	START	7600	TADDR	709F	VN	7602	WARMS	7103
XSAVE	7604								

```
                     ***********************************
                     *                                 *
                     * LOCATE, a utility command program *
                     *   for FIEX to find the beginning  *
                     *   and ending addresses in memory, *
                     *   plus the program starting addr.,*
                     *   of a binary file on disk.       *
                     *   The program is based on "GET"   *
                     *   and "FIIE LOADER" routines in   *
                     *   FIEX, modified.                 *
                     *   Dec. 8, 1978, by R.L.Pigford    *
                     ***********************************

                              NAM     LOCATE
                     *
                     * System equates
                     *
7118                 PSTRING  EQU     $7118   cr/lf + print string
711E                 PCRLF    EQU     $711E
E0C8                 OUT4HS   EQU     $E0C8
758A                 OPEN     EQU     $758A
70B3                 SCRATCH  EQU     $70B3
7103                 WARMS    EQU     $7103   return to FIEX
7525                 FMS1     EQU     $7525
709F                 TADDR    EQU     $709F   transfer address in FIEX
71BD                 ERROR    EQU     $71BD
                     *
                     *
7600                          ORG     $7600
7600  20  04         START    BRA     INIT1
7602  01             VN       FCB     1       version 1 of program
7603                 FLAG     RMB     1
7604                 XSAVE    RMB     2
7606  BD  711E       INIT1    JSR     PCRLF   start with cr/lf
7609  86  00         INIT     LDA A   #0
760B  BD  758A                JSR     OPEN    get file spec.; open for read
760E  25  07                  BCS     RETURN  return to FLEX if through
7610  &C  70B3                INC     SCRATCH
7613  8D  0D                  BSR     LOADER  go to subroutine to read data
7615  20  F2                  BRA     INIT
7617  F6  70B3       RETURN   LDA B   SCRATCH
761A  27  03                  BEQ     JUMP
761C  7E  7103                JMP     WARMS   return to FIEX
761F  7E  71BD       JUMP     JMP     ERROR
                     *
                     * Abbreviated binary loader routine to count memory
                     *
7622  7F  7603       LOADER   CLR     FLAG    set flag to permit output of
7625  BD  7525       LDR1     JSR     FMS1     start addr.; get first byte
7628  81  02                  CMP A   #2      is it start-of-record indicator?
7629  27  31                  BEQ     LDR2    branch if yes
762C  81  16                  CMP A   #$16    is it transfer-addr. indicator?
762E  26  F5                  BNE     LDR1     if no, return for another byte
7630  CE  76AF                LDX     #MSG2   point to end-mem. message
7633  BD  7118                JSR     PSTRING print end addr. message
7636  FE  7604                LDX     XSAVE   get end mem. addr.+1
7639  09                      DEX             correct it
763A  FF  7604                STX     XSAVE   save it
763D  CE  7604                IDX     #XSAVE  point to end mem. addr.
7640  BD  E0C8                JSR     OUT4HS  print it
```

```
7643  BD  7525              JSR     FMS1     get transfer addr., msb
7646  B7  709F              STA A   TADDR
7649  BD  7525              JSR     FMS1     get transfer addr., lsb
764C  B7  70A0              STA A   TADDR+1
764F  CE  76C6              LDX     #MSG3    point to starting addr. msg.
7652  BD  7118              JSR     PSTRING  print it
7655  CE  709F              LDX     #TADDR
7658  BD  E0C8              JSR     OUT4HS   print starting address
765B  20  C8                BRA     LDR1
765D  BD  7525      LDR2    JSR     FMS1     get begin addr., msb
7660  36                    PSH A
7661  BD  7525              JSR     FMS1     get begin addr., lsb
7664  33                    PUL B
7665  B7  7605              STA A   XSAVE+1  store lsb
7668  F7  7604              STA B   XSAVE    store msb
766B  7D  7603              TST     FLAG     test for first time through
766E  27  02                BEQ     LDR5     branch if yes for print
7670  20  0C                BRA     LDR4     skip print
7672  CE  7698      LDR5    LDX     #MSG1    point to begin addr. msg.
7675  BD  7118              JSR     PSTRING
7678  CE  7604              LDX     #XSAVE   point to begin addr.
767B  BD  E0C8              JSR     OUT4HS   print addr.
767E  86  01        LDR4    LDA A   #1       set flag to suppress begin-
7680  B7  7603              STA A   FLAG        addr. print
7683  BD  7525              JSR     FMS1     get no. data bytes in sector
7686  16                    TAB              store it in B-reg.
7687  27  9C                BEQ     LDR1
7689  BD  7525      LDR3    JSR     FMS1     get a data byte
768C  FE  7604              LDX     XSAVE    get byte counter
768F  08                    INX              increment it
7690  FF  7604              STX     XSAVE    save new value
7693  5A                    DEC B            reduce word count
7694  26  F3                BNE     LDR3     return for next word
7696  20  8D                BRA     LDR1     return for next sector
                    *
                    * output strings
                    *
7698  50            MSG1    FCC     /PGM. STORAGE BEGINS AT /
76AE  04                    FCB     4
76AF  50            MSG2    FCC     /PGM. STORAGE ENDS AT   /
76C5  04                    FCB     4
76C6  53            MSG3    FCC     /STARTING ADDRESS  =    /
76DC  04                    FCB     4
                    *
                            END     START

NO ERROR(S) DETECTED
```

A 20 MA PRINTER WITH SWTPC

Terry Perdue
1470 Wilson Rd.
St. Joseph, MO 49085

Here is a convenient way to interface a 20 mA. printer, such as the Model 33, to a SWTPC computer. It is assumed that the system uses the MP-C control board to interface an RS-232 video terminal.

The jumper installed between holes TI and TC on the MP-C board is

removed, and the printer is connected to holes TI (keyboard), TO (print mechanism), and TC (common). (If the printer has no keyboard, leave the jumper installed.) The printer may be used simultaneously with the video terminal, and placed in local mode when hard copy is not desired.

It is often useful to be able to disable output to the printer during the course of a program, for instance when prompts are needed. The accompanying program NOPRNT allows two ways of accomplishing this. Three subroutines are provided that are identical to INEEE, OUTEEE and PDATA1, except that characters will appear only on the CRT.

Alternatively, the PTRON and PTROFF subroutines may be used alone to enable and disable the printer, respectively. Note that the printer's keyboard is not disabled by PTROFF, and will be operative whenever the printer is on-line.

For these routines to operate, a simple hardware modification is required to the MP-C board. This modification is detailed in the listing. It is simple to make, and in no way affects normal system operation.

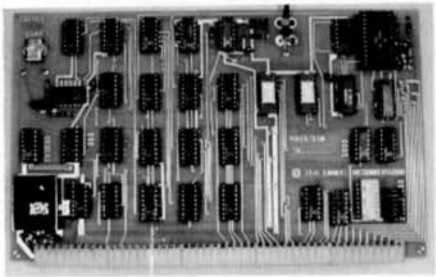
These routines may be assembled to reside anywhere desired, whether in ROM or as part of a user program.

```
                         NAM     NOPRNT


                * PRINTER IS CONNECTED TO 20 MA. SERIAL OUTPUT
                * OF MP-C CONTROL INTERFACE, (PINS TO AND TC).
                * MP-C BOARD MODIFIED AS FOLLOWS:
                * (1)  LIFT PIN 1 OF IC-6, AND JUMPER TO PIN
                *      13 (PB3) OF THE PIA, IC-1.  INSTALL
                *      10K PULLUP BETWEEN THIS POINT AND +5V.
                * (2)  MOVE END OF R4 FROM IC-5, PIN 11, TO
                *      IC-6, PIN 2.
                * SINCE PB3 IS NORMALLY PROGRAMMED AS AN INPUT,
                * IT IS IN TRI-STATE, AND THE PULL-UP ON PIN 1
                * OF IC-6 ENABLES THAT GATE TO ALLOW NORMAL
                * PRINTER OPERATION.  WHEN NINCH, NOUTCH OR
                * NOUTST IS CALLED, THE PTROFF ROUTINE TAKES
                * PB3 LOW, DISABLING PRINTER OUTPUT.

                         OPT     O

                * EXTERNAL EQUATES

      E1AC      INEEE  EQU   $E1AC    CHARACTER INPUT
      E1D1      OUTEEE EQU   $E1D1    CHARACTER OUTPUT
```

```
      E07E    PDATA1 EQU    $E07E    CHAR. STRING OUTPUT
      8006    PIADB  EQU    $8006    MP-C PIA, SIDE B
      A04A    XTEMP  EQU    $A04A    TEMP. STORAGE FOR X

0000                 ORG    0

                     ***
                     * NON-PRINTING SUBSTITUTE FOR INEEE
                     * JSR OR BSR HERE TO INPUT A CHAR.
                     ***
0000 8D 21    NINCH  BSR    PTROFF   DISABLE PRINTER
0002 BD E1AC         JSR    INEEE    CHARACTER INPUT
0005 20 0C           BRA    PTRON    ENABLE PRINTER & RTN
                     ***
                     * NON-PRINTING SUBSTITUTE FOR OUTEEE
                     * JSR OR BSR HERE TO OUTPUT A CHAR.
                     ***
0007 8D 1A    NOUTCH BSR    PTROFF   DISABLE PRINTER
0009 BD E1D1         JSR    OUTEEE   CHARACTER OUTPUT
000C 20 05           BRA    PTRON    ENABLE PRINTER & RETURN
                     ***
                     * NON-PRINTING SUBSTITUTE FOR PDATA1
                     * JSR OR BSR HERE TO OUTPUT A STRING
                     ***
000E 8D 13    NOUTST BSR    PTROFF   DISABLE PRINTER
0010 BD E07E         JSR    PDATA1   OUTPUT STRING
                     ***
                     * JSR OR BSR HERE TO RE-ENABLE PRINTER
                     ***
0013 FF A04A  PTRON  STX    XTEMP    SAVE X
0016 7F 8007         CLR    PIADB+1  TO ADDRESS DDRB
0019 CE 0734         LDX    #$734
001C FF 8006         STX    PIADB    PB3 = INPUT (TRI-ST.) AGAIN
001F FE A04A         LDX    XTEMP    RESTORE X
0022 39              RTS             RETURN TO CALLING PROGRAM
                     ***
                     * JSR OR BSR HERE TO DISABLE PRINTER
                     ***
0023 FF A04A  PTROFF STX    XTEMP    SAVE X
0026 36              PSH A           SAVE CHAR. TO BE OUTPUT
0027 7F 8007         CLR    PIADB+1  TO ADDRESS DDRB
002A CE 0F34         LDX    #$F34
002D FF 8006         STX    PIADB    MAKE PB3 AN OUTPUT
0030 FE A04A         LDX    XTEMP    RESTORE X
0033 86 F7           LDA A  #$F7
0035 B4 8006         AND A  PIADB
0038 B7 8006         STA A  PIADB    FORCE PB3 LOW
003B 32              PUL A           RETRIEVE CHAR.
003C 39              RTS

                     END

TOTAL ERRORS 00000


ENTER PASS : 1P,2P,2L,2T
```

A VIDEO BOARD & IMPROVED MONITOR
for the S-50 BUS

J.L. Pentecost
3605 Clubwood Tr.
Marietta, GA 30067



Among the video boards for the S-50 buss, the Thomas Instrumentation unit* Figure 1, offers an economical terminal capability for the experimenter. This board can be implemented for less than $100 and has excellent capabilities. For those unfamiliar with this video board, it provides a switch selectable 1K RAM memory which can be addressed by the CPU and is also used to refresh the video screen simultaneously. The screen format is 16 lines of 64 characters. The 1K RAM may be placed anywhere in memory with the on-board DIP switch. Interesting views of the memory contents during symbol table sorts and calculations can be obtained by using the 1K video memory to overlay program RAM location where the action is occuring.

While the Thomas Instrumentation video display is not "glitch-less" when the memory is addressed, the minor "noise" on the screen when the display is changed is not objectionable. The video display operates at an equivalent baud rate of about 17K baud while scrolling. Obviously, single page write times are much faster, since the entire page must be rewritten in memory for each added line while scrolling. Single page write rates are approximately 50,000 characters/sec or 20 MS per page, so updating the video display is rapid!

For monitor use, the Video RAM can be placed at any convenient spot. CØØØ-C3FF HEX was selected, though 8ØØØ-9ØØØ HEX could have been used also, if complete addressing of the I/O were implemented. Suitable output routines are provided by Thomas Instrumentation, but for complete cursor control (cursor right and cursor up) minor additions to these routines are required. These output routines require only about 200 bytes of memory, so they can be easily placed in EPROM Memory on the SWTPC MPA2 board. It is also necessary to use 6 bytes of AØØØ RAM memory to store the video pointer, 2 flags and the blink time count for terminal - type operation of the video RAM board. For convenience, the OUTEEE routine at E1D1 should be modified to include these output routines, and a keyboard, to substitute for the terminal, should also be incorporated for the INEEE (E1AC) routine. Thus several revisions of the SWTBUG minitor listings are necessary.

This article also concerns the development of a SWTBUG compatible monitor (with expanded functions) which also contains the routines to drive the Thomas Instrumentation Video RAM board. The entry points for all MIKBUG/SWTBUG subroutines have been maintained and many of the commands have the same letter, though the input of the address data is slightly modified. The features desired in this monitor were:

a. compatibility with SWTBUG

b. Improved input and output switching by software control

c. Additional commands in the monitor for often-needed functions.

d. Video routines and independent keyboard routines to use the system without an external terminal.

e. Complete functioning of the monitor, without the video board or keyboard I/D card, if an external

monitor is used.

f. Operation of the monitor, from the independent keyboard, even though output or input was expected at a port where no card was present.

All of these objectives have been met, and the monitor commands, with a brief functional description, are given in TABLE I. Flow charts for the input and output routines are given in Figures 2 and 3. The character input routine was written with a separate BREAK subroutine which polls for input and returns with the carry set if any input is ready at the port or keyboard. This can be substituted for similar routines in various games and basic versions which often require modification when the I/O is changed. As can be seen, the I/O routines are rather complex because (a) there are added features; (b) they run two I/O "ports" simultaneously, independently; and (c) they automatically switch from AClA to PlA as the port address is changed. Some of these added monitor functions are available with programs which operate through FLEX, but for other programs and tape systems, these added functions are invaluable.

As an example, a printer can be put at any port and the output vectored to that port, in addition to appearing on the video board minitor. The printer can be turned on and off by the "Q" command. The output can also be temporarily interrupted by a control S command. This also allows printing of basic program output at a selected printer port, even though the basic program was not written to output through that port.

The monitor operates both serial and parallel I/O, depending on port selection. The 4 ports from $8ØØØ through $8ØØC are assumed to be AClA ports - no parallel card is allowed at port #1.

All ports above 8ØØC are assumed to be PlA ports by the monitor. The B side of Port #7 was selected for the keyboard input since Part #7 is normally used for parallel output through the A side. Thus with this keyboard at port #7, and the video board, the entire 6800 system can be operated without an external terminal.

Operation without a keyboard and without the video board, requires an external terminal at port #1. When operated in this configuration, the monitor supports all commands except the display of control characters. All video and keyboard functions are ignored. If both a terminal at Port #1 and the video board/ keyboard are used, the same display and control functions are maintained on both terminals independently. If the AC-30 is connected at Port #1, with or without a terminal, the operation is compatible with that of other monitors. The AC-30 can also be placed at any AClA Port if that port is selected with the monitor "O" command.

Complete HEX format and binary type routines have been included in the monitor. This allows loading and generation of both type tapes with a suitable tape interface.

Other monitor routines are included in the monitor to ease software development and patching. Among these are the 2 byte address search, breakpoint insertion, and block memory move commands. The "I" command is the "Thpmpson Lister" program Ref. to display instructions as 1, 2 or 3 bytes, as appropriate. This display is most useful in checking to see if a program is correctly entered or that a change is made properly. Disk operations are supported with the jump to FLEX at $7103 and disk boot routine. For FLEX 2.0, residing at AØØØ, this jump address is easily modified.

Terminal-less operation has one drawback; remote terminal operation with a modem is often desired and a terminal can be used with both

modem and CPU. One approach to providing an equivalent "terminal" for use with a modem using the video board is a short monitor routine which converts the CPU/video board/key board system into a "dumb" terminal, (the "X" command). The serial output port and serial input data is displayed on the video board monitor, just as though it were a terminal.

The cursor display is caused to blink by software timing loop, but this loop is only active while the CPU is polling for input at E1AC, hence there is no constant CPU overhead to blink the cursor. The cursor is norma-ly on and not blinking while the CPU is performing tasks not involving E1AC input. This implementation of a blinking cursor provides a reasonable compromise between visibility and CPU usage.

One word of caution is appropriate. The AØØØ RAM usage to support this monitor includes dedication of locations $AØ14 through $AØ19 for monitor use. These added bytes are used with the video and I/O routines routines and are essential. Any attempt to use these bytes in memory diagnostics or for other purposes will fail. All other usage of the $AØØØ RAM is identical to that of SWTBUG.

This entire monitor requires about 1600 (decimal) bytes and is easily programmed in a single 2716 EPROM (with 430 bytes free for other routines) which can be used on the MPA2 CPU board to replace SWTBUG. The complete commented source listings for this monitor are 16 pages and too lengthy to include in an article of this type, but are available from the author.*

In summary an approach to a compact, economical, single unit 6800 system is described. A lower cost system results, the display speed is increased and less disk top space is required. Improved monitor functions are also provided for both program development and system expansion.

* Comment Source Listing of JOEBUG Monitor $5.00; Copy of Object Code on supplied tape or disk or EPROM $5.00.

```
                         *START INPUT ROUTINE
E1A5 8D 07               INLOOP   BSR     BREAK      CK FOR INPUT
E1A7 25 68                        BCS     INCHA      GET CHARACTER
E1A9 BD E2 3D                     JSR     BLINK      CURSOR
E1AC 20 F7               INEEE    BRA     INLOOP     GO TO MAIN LOOP
                         *BREAK TEST ROUTINE
E1AE 8D 55               BREAK    BSR     SAVGET     GET PORT ADDR &TYPE
E1B0 2C 7C                        BGE     PIACK      ITS A PIA
E1B2 A6 00               ACIACK   LDA A   0,X        GET STATUS
E1B4 4C                           INC A              FF=00
E1B5 27 0A                        BEQ     KEYCK      NO CARD IN PORT
E1B7 4A                           DEC A              RESTORE A
E1B8 47                           ASR A
E1B9 24 06                        BCC     KEYCK      NO INPUT
E1BB 4F                  RTN3     CLR A              INPUT FROM PORT
E1BC 0D                           SEC                SET FLAG FOR INPUT
E1BD FE A0 10                     LDX     XTEMP      RECOVER X
E1C0 39                           RTS
                         *KEYBOARD CHECK ROUTINE
E1C1 B6 80 1F            KEYCK    LDA A   $801F      GET STATUS
E1C4 4C                           INC A              FF=00
E1C5 27 02                        BEQ     RTN2       NO CARD AT PORT
E1C7 2B 03                        BMI     SETFLG     MARK INPUT
E1C9 0C                  RTN2     CLC                NO INPUT
E1CA 20 F1                        BRA     RTN3+2     GET X, GO
E1CC 8D ED               SETFLG   BSR     RTN3       SET INPUT FLAG
E1CE 8A 80                        ORA A   #$80       MARK FOR KEYBOARD INPUT
E1D0 39                           RTS                DONE
                         *OUTPUT ROUTINE, A REGISTER TO PORT & SCREEN
E1D1 36                  OUTEEE   PSH A
E1D2 8D DA                        BSR     BREAK      CK FOR INTERUPT
E1D4 24 08                        BCC     CNTX       CONTINUE
E1D6 8D 39                        BSR     INCHA      GET INPUT
E1D8 81 13                        CMP A   #$13       IS IT CTL S ?
E1DA 26 02                        BNE     CNTX       NO, CONTINUE
E1DC 8D CE                        BSR     INEEE      WAIT THERE
E1DE 32                  CNTX     PUL A              IGNORE INPUT
E1DF 36                           PSH A
E1E0 8D 74                        BSR     VIDEO      PUT ON SCREEN
E1E2 7D A0 17            OUTSW    TST     PROUT      PUT IT OUT TO PORT ?
E1E5 26 11                        BNE     RTN5       NO GO
E1E7 8D 1C                        BSR     SAVGET     GET PORT ADDR & TYPE
E1E9 32                           PUL A              GET A
E1EA 37                           PSH B              SAVE B
E1EB 2C 0D                        BGE     PIAOUT     ITS A PIA
E1ED E6 00               ACIAOU   LDA B   0,X        GET STATUS
E1EF 57                           ASR B
E1F0 57                           ASR B              FLAG TO CARRY
E1F1 24 FA                        BCC     ACIAOU     NOT READY
E1F3 A7 01                        STA A   01,X       SEND IT OUT
E1F5 33                  RTN4     PUL B              RESTORE B
E1F6 20 C5                        BRA     RTN3+2     RESTORE X, GO
E1F8 32                  RTN5     PUL A              GET A, CORRECT STACK
E1F9 39                           RTS                GO
E1FA E6 03               PIAOUT   LDA B   03,X       GET STATUS
E1FC 5C                           INC B
E1FD 27 F6                        BEQ     RTN4       NO I/O CARD
E1FF 2A F9                        BPL     PIAOUT     NOT READY
E201 A7 02                        STA A   02,X       SEND IT OUT
E203 20 F0                        BRA     RTN4       DONE
E205 FF A0 10            SAVGET   STX     XTEMP      SAVE X
E208 FE A0 0A                     LDX     PORADD     GET PORT
E20B B6 A0 0B                     LDA A   PORADD+1   GET LSBYTE
E20E 81 10                        CMP A   #$10       IS IT A PIA ?
```

```
E210 39                 RTS
E211 4D        INCHA    TST A              CK IF MSB IS SET
E212 2B 15              BMI    KEYIN       GO TO KEYBOARD
E214 8D FF              BSR    SAVGET      GET PORT
E216 2C 04              BGE    PIAIN       ITS A PIA
E218 A6 01     ACIAIN   LDA A  01,X        GET INPUT
E21A 20 02              BRA    NULOUT
E21C A6 00     PIAIN    LDA A  0,X         GET INPUT
E21E FE A0 10  NULOUT   LDX    XTEMP       RECOVER X
E221 84 7F     NUL1     AND A  #$7F        7 BITS ONLY
E223 81 7F              CMP A  #$7F        A RUBOUT ?
E225 27 16              BEQ    BLINK       GO, NO INPUT
E227 20 0E              BRA    ECHOCK      ECHO ?
E229 B6 80 1E  KEYIN    LDA A  $801E       GET DATA
E22C 20 09              BRA    ECHOCK      ECHO IT?
E22E A6 01     PIACK    LDA A  01,X        GET STATUS
E230 4C                 INC A              FF=00
E231 27 8E              BEQ    KEYCK       NO I/O CARD
E233 2A 8C              BPL    KEYCK       NO INPUT
E235 20 84              BRA    RTN3        INPUT, SET FLAG
E237 7D A0 0C  ECHOCK   TST    PORECH      ECHO FLAG
E23A 27 95              BEQ    OUTEEE      ECHO
E23C 39                 RTS                NO ECHO
E23D 7A A0 19  BLINK    DEC    BLNKTM+1    DECREMENT TIME
E240 26 13              BNE    RTNY        CK FOR BORROW
E242 7A A0 18           DEC    BLNKTM      CORRECT BORROW
E245 26 0E     BLNK1    BNE    RTNY        TIME UP? NO,RETURN
E247 FF A0 10           STX    XTEMP       SAVE IT
E24A CE 08 00           LDX    #$0800      YES, RESET TIMER
E24D FF A0 18           STX    BLNKTM      SAVE TIME COUNT
E250 8D 3D              BSR    CURSOR      CHANGE CURSOR
E252 FF A0 10           LDX    XTEMP       RESTORE X
E255 39        RTNY     RTS                GO
               *VIDEO OUTPUT ROUTINE PUTS A ON SCREEN (A CHANGED)
E256 81 0D     VIDEO    CMP A  #$0D        CHECK FOR CARRIAGE RETN
E258 27 42              BEQ    DOCR
E25A 81 0A              CMP A  #$0A        CHECK FOR LINE FEED
E25C 27 4A              BEQ    DOLF
E25E 81 01              CMP A  #$01        CHECK FOR HOME-UP (CNTL A)
E260 27 5C              BEQ    DOCNTA
E262 81 16              CMP A  #$16        CHECK FOR CLEAR TO END (CNTL V)
E264 27 5F              BEQ    DOCNTV
E266 81 08              CMP A  #$08        CHECK FOR BACKSPACE (CNTL H)
E268 27 6D              BEQ    DOCNTH
E26A 81 17              CMP A  #$17        CHECK FOR UP CURSOR (CNTL W)
E26C 27 76              BEQ    DOUP
E26E 81 13              CMP A  #$13        CHECK FOR RIGHT CUR (CNTL S)
E270 27 74              BEQ    DORT
E272 7D A0 16           TST    PRCTL       CHECK WHETHER TO DISPLAY CTL CHAR
E275 26 05              BNE    VOUT        FLAG ON, PUT OUT ALL CTL CHAR
E277 81 20              CMP A  #$20        CHECK FOR OTHER CONTROL CHAR
E279 2A 01              BPL    VOUT        PUT OUT NON CNTL CHAR,IGNORE OTHER

E27B 39                 RTS
E27C FF A0 10  VOUT     STX    XTEMP       SAVE INDEX REG
E27F FE A0 14           LDX    VIDPNT      GET POINTER
E282 A7 00              STA A  0,X         PUT OUT CHAR
E284 08                 INX                GET NEXT CHAR POSN
E285 8C C4 00           CPX    #TOPSCR+1024 TO END OF PAGE?
E288 26 02              BNE    VIDOT1      IF NOT, SKIP SCROLL
E28A 8D 69              BSR    DOROLL      SCROLL SCREEN UP ONE
E28C FF A0 14  VIDOT1   STX    VIDPNT      SAVE POINTER
               *START CURSOR ON ROUTINE
E28F FE A0 14  CURSOR   LDX    VIDPNT      GET POINTER
```

```
E292 A6 00                LDA A   0,X         GET CHARACTER AT X
E294 88 80                EOR A   #$80        FLIP REVERSE BIT
E296 A7 00                STA A   0,X         REPLACE REVERSED CHAR
E298 FE A0 10             LDX     XTEMP       RESTORE X
E29B 39                   RTS                 RETURN
                  *CARRIAGE RETURN ROUTINE
E29C 8D 4A        DOCR    BSR     RESTOR      TURN OFF CURSOR
E29E B6 A0 15             LDA A   VIDPNT+1    GET LS BYTE
E2A1 84 C0                AND A   #$C0        MAKE LS BITS 0
E2A3 B7 A0 15             STA A   VIDPNT+1
E2A6 20 E7                BRA     CURSOR      TURN ON CURSOR AND RETURN
                  *LINE FEED ROUTINE
E2A8 8D 3E        DOLF    BSR     RESTOR      TURN OFF CURSOR
E2AA 86 40                LDA A   #64         GO 64 CHARACTERS FORWARD
E2AC 08           DOLF1   INX                 PUSH CURSOR POSN FORWARD ONE
E2AD 8C C4 00             CPX     #TOPSCR+1024 PAST END YET?
E2B0 27 05                BEQ     DOLF2       IF PAST, SCROLL AND CONTINUE
E2B2 4A                   DEC A               DECREMENT A ONE
E2B3 26 F7                BNE     DOLF1       IF A IS NOT 0, CONTINUE
E2B5 20 D5                BRA     VIDOT1      SAVE NEW CURSOR POSN,TURN ON
E2B7 36           DOLF2   PSH A               SAVE COUNTER
E2B8 8D 3B                BSR     DOROLL      SCROLL SCREEN UP AND CLEAR A LINE
E2BA 09                   DEX                 CORRECT POINTER FOR RE-ENTRY
E2BB 32                   PUL A               RESTORE COUNTER
E2BC 20 EE                BRA     DOLF1       CONTINUE FORWARD MOVE
                  *HOME-UP ROUTINE
E2BE 8D 28        DOCNTA  BSR     RESTOR      TURN OFF CURSOR
E2C0 CE C0 00             LDX     #TOPSCR     POINT TO TOP OF SCREEN
E2C3 20 C7                BRA     VIDOT1      CLEAN UP
                  *CLEAR TO END OF SCREEN ROUTINE
E2C5 FF A0 10     DOCNTV  STX     XTEMP       SAVE X
E2C8 FE A0 14             LDX     VIDPNT
E2CB 86 20                LDA A   #$20        GET A SPACE
E2CD A7 00        DOCNV1  STA A   0,X         PUT IN A SPACE
E2CF 08                   INX                 MOVE TO NEXT POSN
E2D0 8C C4 00             CPX     #TOPSCR+1024 TO END YET?
E2D3 26 F8                BNE     DOCNV1      NO, CONTINUE
E2D5 20 B8                BRA     CURSOR      PUT ON CURSOR AND RETURN
                  *BACKSPACE ROUTINE
E2D7 8D 0F        DOCNTH  BSR     RESTOR      TURN OFF CURSOR
E2D9 09                   DEX                 BACK UP ONE
E2DA 8C BF FF             CPX     #TOPSCR-1   CHECK FOR WRAP-AROUND
E2DD 26 AD                BNE     VIDOT1      NO WRAP, ALL DONE
E2DF CE C3 FF             LDX     #TOPSCR+1023 CORRECT WRAP
E2E2 20 A8        VIDOT2  BRA     VIDOT1      CLEAN UP AND GO
                  *BRIDGES FOR RELATIVE BRANCHES
E2E4 20 2D        DOUP    BRA     DOUP1
E2E6 20 3F        DORT    BRA     DORT1
                  *CURSOR OFF ROUTINE
E2E8 FF A0 10     RESTOR  STX     XTEMP       SAVE X REG
E2EB FE A0 14             LDX     VIDPNT      GET CURSOR POSN
E2EE A6 00                LDA A   0,X         GET BYTE THERE
E2F0 84 7F                AND A   #$7F        MASK OFF 7 BITS
E2F2 A7 00                STA A   0,X         REPLACE BYTE
E2F4 39                   RTS
                  *SCROLL UP AND CLEAR LINE ROUTINE
E2F5 CE C0 00     DOROLL  LDX     #TOPSCR     POINT TO TOP OF SCREEN
E2F8 A6 40        DOROL1  LDA A   64,X        GET CHAR ON 2ND LINE
E2FA A7 00                STA A   0,X         PUT CHAR AT POINTER
E2FC 08                   INX                 INCREMENT TO NEXT POSN
E2FD 8C C3 C0             CPX     #TOPSCR+960 CONTINUE TO BOTTOM LINE
E300 26 F6                BNE     DOROL1
E302 86 20                LDA A   #$20        GET A SPACE
```

```
E304 A7 00      DOROL2  STA A   0,X          PUT IT IN PLACE
E306 08                 INX                  INCREMENT TO NEXT POSN
E307 8C C4 00           CPX     #TOPSCR+1024
E30A 26 F8              BNE     DOROL2       CONTINUE TO END OF LINE
E30C CE C3 C0           LDX     #TOPSCR+960  POINT TO LAST LINE
E30F FF A0 14           STX     VIDPNT       RESTORE POINTER
E312 39                 RTS                  RETURN
                *UP CURSOR ROUTINE
E313 8D D3      DOUP1   BSR     RESTOR       TURN OFF CURSOR
E315 86 40              LDA A   #64          SET COUNTER TO 64
E317 09         DOUP2   DEX                  BACK UP ONE
E318 8C BF FF           CPX     #TOPSCR-1    WRAP AROUND?
E31B 27 05              BEQ     WRAP         CORRECT IT
E31D 4A                 DEC A                COUNT OFF ONE
E31E 26 F7              BNE     DOUP2        CONTINUE TILL DONE
E320 20 C0              BRA     VIDOT2       DONE, CLEAN UP AND GO
E322 CE C4 00   WRAP    LDX     #TOPSCR+1024 GO TO BOTTOM
E325 20 F0              BRA     DOUP2        CONTINUE
                *CURSOR RIGHT ROUTINE
E327 8D BF      DORT1   BSR     RESTOR       TURN OFF CURSOR
E329 08                 INX                  GO RIGHT ONE
E32A 8C C4 00           CPX     #TOPSCR+1024
E32D 26 B3              BNE     VIDOT2       DONE,GO
E32F 8D C4              BSR     DOROLL       SCROLL LINE
E331 20 AF              BRA     VIDOT2       DONE
```

TSC BASIC FOR THE 6800

Prepared by: David Shirk — TSC
Technical Report — B68

I. Introduction

In the Fall of 1977, Technical Systems Consultants decided to develop a BASIC interpreter for the 6800 microprocessor. There were several factors motivating this project. Highest on the list was the need for a generally available 6800 BASIC which offered more speed than that provided by the famed Uiterwyk BASIC, the first 6800 BASIC available. An excellent BASIC was available from Microsoft at the time but was priced to scare away most hobbyists and casual users. Another prompting factor was a series of articles in "Kilobaud" magazine which compared the various microcomputer BASIC's in speed (see references 1-3). In one of these articles (ref. 2), Bill Gates of Microsoft was quoted as saying "the 6502 is an inherently faster processor". This statement was apparently made because the Microsoft 6502 BASIC was currently the fastest they had written to that date. TSC has used the 6800, 8080, and 6502 microprocessors and has found no significant speed differences among them. In certain applications one may outperform the others, but these are usually specific instances. The 6800 BASIC was hopefully going to prove that the important speed factor is efficiency of the software and not the characteristics of the 3 micros mentioned.

Another remark along the same lines which we found troublesome was made by J. G. Letwin of Heath Company (see ref. 3). In comparing the Heath H8 and H11 computers running BASIC, the H11 came out on top by almost a factor of two. The comment was made that the "Heath H8 computer, an 8080A machine, performed considerably slower than did the H11 systems, for obvious reasons". These reasons were not "obvious" to us since the LSI-11 processor, even though a 16 bit machine, is not extremely fast. Again, TSC felt that a 6800 BASIC could be developed which would outperform the H11 BASIC (Note: We did not consider competing with the LSI-11 equipped with its floating point firmware since the same option is not available for the 6800).

The BASIC was developed over a fourteen month period. All of the original goals were met with the exception of one, the overall size of the program. The target was an interpreter which required 8K to 8.5K of memory. The final product is 9.5K, a value which is still very reasonable for the number of features supported. It was decided that in the never ending trade-off between size and speed of a program, speed should be the winning factor. This decision was made because memory is becoming less and less expensive, while the cost of CPU speed is somewhat fixed. The speed of the TSC 6800 BASIC exceeds all other currently available floating point BASIC's for 8 bit micros. The goal of outperforming the LSI-11 BASIC as supplied by the Heath Company was also achieved. All of the pertinent timing information will follow shortly.

The resultant BASIC is very easy to use and is compatible with the majority of BASIC's on the market. A full disk file version for the FLEX™ operating system will be available in the near future, as well as a 6809 version. The 6809 BASIC will show an even more dramatic improvement in speed and will be available in May, 1979!

II. Features of TSC BASIC

The TSC BASIC for the 6800 supports all of the standard BASIC statements and functions. Many extended features have been provided as well. A complete list of features appears in the appendix. Some of the highlights will be described here. One nice feature is the ability to use two letter variable names. Standard BASIC requires either single letter or letter followed by a number. TSC BASIC will allow the use of letter followed by letter which permits limited variable name mnemonics. Most BASIC's allow multiple statements per line. Some use a colon (':') as the separater while others use a backslash ('\'). TSC BASIC supports both, thus aleviating some confusion for those entering published BASIC programs.

A very important new feature provided is the 'IF THEN ELSE' construct. The majority of BASIC's only support the THEN clause. The addition of the 'ELSE' promotes a more structured type programming style, thus improving readability and conciseness of the program. Along with this, the input line buffer has been increased to 127 characters from the typical 72 to accept the longer lines possible with the 'IF THEN ELSE' statements.

'PEEK' and 'POKE' have been widely accepted as useful BASIC features. The big problem in the past has been the necessity of referencing machine memory addresses in decimal, whereas most users think hex when it comes to address locations. The 'HEX' function has been implemented for this application. This unique function will allow a string expression as an argument and return an equivalent decimal number. The string argument is interpreted as a string of hex digits. As an example, to examine the contents of location $8004 (the '$' implies hex), the following statement may be used; PRINT PEEK(HEX("8004")). This is much easier than having to do a manual conversion from hex to decimal before entering the statement! The HEX function also comes in handy when using the logical operators (AND, NOT, and OR) to perform bit wise operations. The mask can be represented as a hex value as opposed to using decimal.

Many of the popular BASIC's put restricting limits on several of the statements, such as only allowing a maximum of 8 nested FOR-NEXT loops. TSC BASIC allows as many nested loops and subroutine calls as memory

will permit. This applies to string variables as well. Strings may be any length (limited to 65,535 because of the limit on memory) and are fully dynamic (their length does not have to be specified before use and may change in size at any time). Array sizes are also limited only by the amount of memory installed in the machine. Both single and double dimensioned arrays are supported and can be floating point or string type variables. It should be noted that subscripts of 0 are permitted in TSC BASIC!

A very convenient string oriented statement has been included, 'INPUT LINE'. This feature allows a complete input line (leading spaces, commas, quotes, etc.) to be assigned to a string variable. This solves the problem of entering certain characters from an INPUT statement. Just a few of the nice features have been described here. For a complete list of commands, statements, functions, and specifications, consult the appendix.

III. Timing Comparisons

The features were presented before the timing information because many times, features are more important. For those who find speed to be the important factor, the following tables will show TSC BASIC to be on top! Table I shows the results of the 7 "Kilobaud" benchmarks (references 1 and 2). Since a 2 megahertz 6502 was listed, the TSC BASIC is also listed at 2 megahertz along with the standard speed. Table II shows the results of the "Heath benchmarks" also presented in "Kilobaud" magazine (reference 3). From these results it can be seen that the TSC BASIC runs faster overall than the Heath H11 computer! Examine these tables closely. They speak for themselves.

IV. Conclusion

TSC succeeded in all of its original goals in implementing the TSC BASIC for the 6800. A wide variety of statements and functions are supported and the speed of the interpreter exceeds that of all other floating point BASIC's for 8 bit micros. Overall, the BASIC is an efficient and easy to use program!

V. References

1. T. Rugg, "BASIC Timing Comparisons", KILOBAUD, June 1977, pp 66-70
2. T. Rugg, "BASIC Timing Comparisons - Revisited", KILOBAUD, Oct 1977, pp 20-25
3. J. Letwin, "Another Look at Benchmark Programs", KILOBAUD, Nov 1977, pp 98-101

TABLE I

BASIC & Machine	CPU	1	2	3	4	5	6	7	Ave.
TSC on SWTPC @ 2Mhz	6800	0.4	1.6	5.0	5.2	5.5	8.5	13.1	5.61
OSI 8K @ 2Mhz	6502	0.9	4.6	8.2	9.3	10.0	14.8	21.6	9.91
TSC on SWTPC @ 1Mhz	6800	0.9	3.2	10.1	10.4	11.0	16.9	26.2	11.24
OSI 8K @ 1Mhz	6502	1.6	8.9	16.2	18.2	19.7	29.2	42.9	19.53
PET BASIC	6502	1.7	9.8	18.6	20.4	22.1	32.6	51.3	22.36
Altair 8K (4.0)	8080	1.7	10.2	21.0	22.5	24.3	36.7	52.4	24.11
TRS 80 Level II	Z80	2.6	10.9	25.4	26.1	31.0	51.0	78.4	32.2
Altair 680B (3.2)	6800	2.5	16.3	30.7	33.4	36.3	55.9	81.8	36.7
HEATH H8	8080	3.5	15.6	31.5	34.5	41.4	67.6	97.4	41.64
Tektronix Level 5	6800	4.8	14.0	33.0	36.3	40.7	68.8	103.8	43.04

Computer Ware BASIC	6800	8.1	16.0	44.6	49.7	53.1	93.0	116.9	54.49
SWTPC BASIC (3.5)	6800	12.8	21.4	82.8	90.5	93.9	149.7	175.7	89.54

TABLE II

Operation	TSC*	TSC	H11	H8
A=B	0.4	0.9	0.5	3.0
B+C	0.4	0.8	1.0	2.3
B-C	0.5	0.9	1.1	2.5
B*C	0.9	1.9	1.9	3.8
B/C	1.2	2.4	2.0	4.9
B↑C	14.3	28.5	23	35
SIN(B)	6.5	13	14	13
COS(B)	7.5	15	16	13
TAN(B)	10.9	21.9	32	24
ATN(B)	8.3	16.5	19	21
LOG(B)	7.7	15.3	14	19
SQR(B)	3.1	6.3	7.2	13
EXP(B)	6.5	13	13	15
5 FOR NEXT	2.4	4.8	7.5	23
A$=B$	0.9	1.9	1.0	4.6
B$+C$	0.5	1.0	2.0	4.5
LEFT$(B$,1)	0.4	0.8	1.7	5.8
A=B(I)	0.2	0.4	0.5	2.6
Totals =	72	145	157	210

*Note: The first column is TSC at 2 Megahertz. Second is at 1 Megahertz.

Ed's Note: Information received as of the date of this article indicate the following may be of some interest to users of SWTPC BASIC, when converting to TSC BASIC.

1. The disk versions will load from disk BASIC source in SWTPC format.

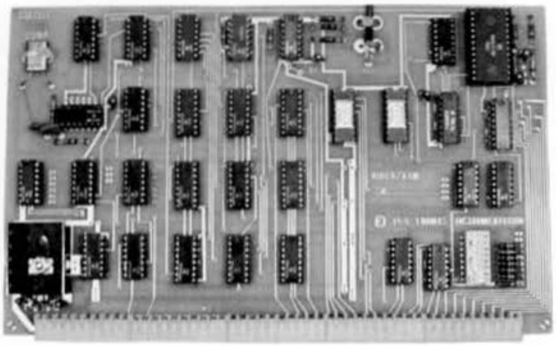
2. The cassette version handles certain delimiters (etc.) in SWTPC BASIC different than is handled in TSC BASIC. This probably means that some SWTPC BASIC developed programs on tape, will not load into TSC BASIC.

3. Coming later is the extended version of TSC BASIC, this version will have extended math precision (12 digits) and some additional commands for business type applications. (PRINT USING, etc.)

4. For those upgrading to the 6809, TSC will have soon the 6809 BASIC. Watch TSC advertising in 68 Micro Journal.

PRICES SHOWN ARE FOR ORDERS RECEIVED BY APRIL 30, 1979

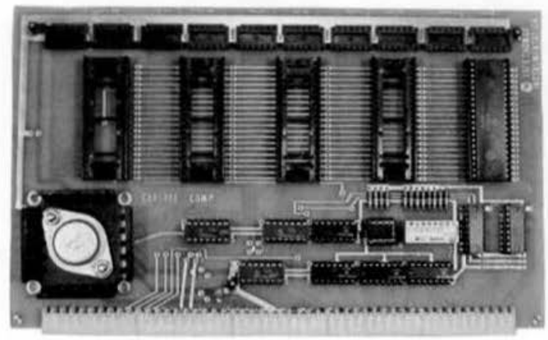
NORMAL PRICES APPROX. 10% HIGHER . . . CALL FOR DEALER, OEM, AND GROUP PRICING



SS-50 VIDEO RAM

- Fully Synchronous Operation to Eliminate Jitter
- Full 128 Set of 7 by 9 Characters & Reverse Image
- 1K of Memory can be Mapped in any 1K Increment
- 10 Pages of Documentation Including Software

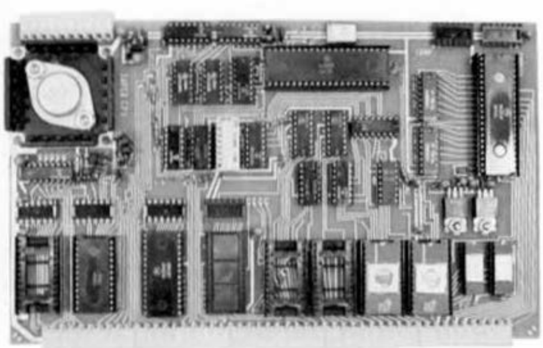
ASSEMBLED $135.00
CARD, CRYSTAL, & DOC $35.00



SS-50 PARALLEL I/O

- 2 Ports Expandable to 10 (Just Add PIA'S @ $6.95 ea.)
- Each Port has 8 I/O Plus 2 Control . . . 100 I/O Lines . . .
- Large Regulator Supplies Power to I/O Connectors
- Board Can be Mapped to any 32 Word Boundary

ASSEMBLED (W/1PIA) $89.00
CARD ONLY $32.00



SS-50 SUPER CPU

- SS-50 Buss . . or . . Stand Alone Computer
- 1K of Ram at $A000, I/O on Board at $A400 (Relocatable)
- 2K Monitor in 2708 Eprom (Expandable to 4K)
- 2 Parallel 8 bit ports with 2 Control Bits and Power
- 1 RS-232 ACIA Port (Expandable to 2nd TTL ACIA)
- 3-16 Bit Counter Timers (Expandable to 6 . . . Add 2nd 6840)
- Additional 128 Byte Ram at 0000 can be Jumpered out
- Battery Back-up Available on 32 Bytes of Ram
- Back/Back with other SS-50 Cards to Make Small Systems

Gimix 16K Without Soft Addressing $298.13
Gimix 16K Static Ram W/Softadd $368.16
Our SS-50 Wire-Rap Card $24.00

3, 4, and 7 slot SS-50 Backplanes
We also Make Non SS-50 Single Board 6800 Sys.

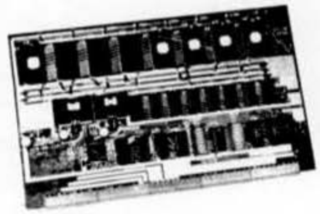
We have been in business for over 8 years, designing industrial machine tool controls and monitors . . . Call us for help with your product or laboratory instrumentation problem . . . we are dealers for:

GIMIX SWTPC SSB

*** * * LET US QUOTE YOU A PACKAGE PRICE * * ***

THOMAS INSTRUMENTATION

2709 Dune Drive, Avalon, N.J. 08202
Phone (609) 967-4280



PSB-08 PROM SYSTEM BOARD

The Micro Works PROM System Board features 1K of high speed (350ns), low power RAM and space for up to 8 2708 EPROMs, both DIP-switch addressable to start on any 8K boundary in memory. The exclusive I/O select feature allows the user to move the I/O locations up to any unused 1K block in the EPROM memory space. This permits memory expansion to a full 56K of contiguous user RAM. An optional +12 volt regulator can be installed for systems incorporating the Smoke Signal Broadcasting PS-1 power supply or its equivalent.

Prices PSB-08 (EPROMs not included) $119.95
PSB-08R (regulated +12) $124.95

NEW PRODUCT

The COMMERCIAL MAILING SYSTEM is a series of programs written in Computerware's RANDOM BASIC for maintaining a mailing list on disk. Since it was written with RANDOM BASIC it provides on-line access and update capability of any record in the file. This system can only be run under Smoke Signal Random Access Disk Operating System. All fields (except date) in the system allow alpha-numeric data. Some of the features of the system include the address label printing that can be directed to any port (0-7), the drive on which the data file resides is operator selectable, optional special code and phone numbers may be maintained, label spacing is operator selectable, a date is maintained with each name (for birthdate, expiration date, etc.), and sorted reports or labels can be requested by a range of values of any one of the following fields: date, code, zip, or name. The facility is provided to maintain and print either a title or a country in the label or report. This is being introduced at $89.95 from COMPUTERWARE 830 First Street Encinitas, CA. 92024. (714) 436-3512

COMPUTERWARE . . .

PRODUCT DESCRIPTION		PRICE
RANDOM ACCESS DISK FILE BASIC		89.95
w/EDIT, line input, & more new features		
SUPER DISK FILE HANDLING BASIC	FLEX ONLY	49.95
w/EDIT, line input, & more new features		
SUPER CASSETTE FILE HANDLING BASIC	cassette	29.95
PROM RESIDENT CASSETTE FILE BASIC	cassette	100.00
(SUPER features)	2716	250.00
PILOT - OUR VERY OWN FOR THE 6800	(SSB or FLEX)	24.95
(with COMMENTED SOURCE LISTING)		37.95
(also with SOURCE CODE ON DISK)		49.95
DISK CHECK FILE MAINT. SYSTEM	(SSB or FLEX)	49.95
* (Complete checking system)		
DISK NAME AND ADDRESS SYSTEM	(SSB or FLEX)	49.95
* (Selective printing - labels)		
(Sorting - Merging - Good manual)		
DISK HOME INVENTORY SYSTEM	SSB ONLY	49.95
* (Random access files - online inquiry)		
(6 Reports either hardcopy or display)		
AMERICANA PLUS (14 NEWTECH Songs)	cassette	15.95
(Dixie, Noel, Eyes of Texas, & more)	(SSB or FLEX)	19.95
CSS FOUR PART #1 (8 NEWTECH Songs)	cassette	24.95
(O'Holy Night, Moonlight Sonata,	(SSB or FLEX)	24.95
Rhapsody in Blue, & more)		
BASIC PGMS #1 (Bluff, Chase, Animal)	cassette	19.95
(Hamurabi, Biorythm, Horse race)	(SSB or FLEX)	19.95
* (Mastermind, State Caps, Skydiver)		
BASIC PGMS #2 (Decision, Black Jack)	cassette	19.95
(Math Lesson, Lunar, Keno)	(SSB or FLEX)	19.95
* (Stockmarket, Furs, Crash, Wumpus)		
BASIC PGMS #3 (Doall, Maze, Roadrace)	cassette	19.95
(Depth charge, Lifetime, Baseball)	(SSB or FLEX)	19.95
* (Interest & Income prop calc)		
WHIZ (TM) - High Speed Binary	cassette	15.95
Cassette Punch and Load System		
DOODLEBUG - EXTENDED SMARTBUG MONITOR	cassette	19.95
* (with SMARTBUG in EPROM)	2716	69.95

* indicates that source listing is included

****** Cassette format: AC-30 --- 8" disk add $2.00 ******
****** ALL SSB Software is available on 8" diskettes ******

C O M P U T E R W A R E   S O F T W A R E   S E R V I C E S
830 First Street   Encinitas, California   92024

PRODUCT DESCRIPTION			PRICE
LEARN ASSEMBLER - PART I		cassette	19.95
(5 Lessons - Requires 16K)		(SSB or FLEX)	19.95
LEARN BASIC PACKAGE (I II & III)		cassette	39.95
(All 12 lessons - need 16K)		(SSB or FLEX)	39.95
RENBAS - RENUMBER BASIC Programs		cassette	
Same features as below		w/source list	24.95
RENBAS - RENUMBER BASIC Programs		(SSB or FLEX)	
Selective starting number		w/source list	24.95
and increment value		w/source disk	34.95
XREF - Assembly Program Label		(SSB or FLEX)	
Cross-Reference - Complete		w/source list	24.95
Commented Source Listing		w/source disk	34.95
SUBMIT - DOS Batch Processor		BFD-68 ONLY	
Macro capabilities		w/source list	24.95
Dos Error Recovery		w/source disk	34.95
BFD-68 DISK TRANSIENTS #1		BFD-68 ONLY	
Includes: Listp, Init, Mem		w/source list	24.95
Lodhex, Savlod, Printp		w/source disk	34.95
Cordmp, Disasm, Convrt, Epend			
BFD-68 DISK TRANSIENTS #2		BFD-68 ONLY	
Includes: Print, Dir, Pip		w/source list	24.95
Cksm, Search, Fill, Filcom		w/source disk	34.95
Delete - Wild Card Options!			
BFD-68 DISK TRANSIENTS #3		BFD-68 ONLY	
Includes: Fdump, Sdump, Map,		w/source list	24.95
Scout, Memdump, Compare,		w/source disk	34.95
Hardcopy for other transients			
FLEX UTILITY COMMANDS #1		FLEX ONLY	19.95
Includes: Examine, Findhex1, 2, 3		w/source disk	24.95
Dir, Kill, Files, Frags, Repeat			

SMOKE SIGNAL BROADCASTING

SMARTBUG		documentation	19.50
If documentation purchased:		(2708)	39.95
		(2716) 5v	49.95
EDITOR or ASSEMBLER	SE-1 OR SA-1	cassette/disk	29.00
EDITOR and ASSEMBLER	SE-1 & SA-1	cassette/disk	53.00
TEXT PROCESSOR		disk	39.95
SSB BASIC AND RANDOM DOS		disk	39.95
TRACE - DISASSEMBLER	TD-1	disk	25.90
		cassette	19.95
SOURCE GENERATOR	SG-1	disk	29.90
		cassette	24.95

COMPUTERWARE...

THE MICRO WORKS

INNOVATIVE PRODUCTS

DESIGNED WITH THE 6800 USER IN MIND!

The folks at the Micro Works would like to say Thanks to the 6800 owners of the world for their enthusiastic response to our products. To do so, we are going to continue to offer our exceptional value 2708 EPROM System with a full $25 discount off list price, to those who buy the complete system, and tell us that they heard about it in Micro 68. The full system includes one of our very popular PSB-08 Prom System Boards, a B-08 Prom Burner, with driver software in 2708, and a L. S. Engineering Prom Eraser. That is EVERYTHING you need to put your system software in EPROM, and change it when you need to. The separate items list for a total of $299.80, but the complete package price to Micro 68 readers is $274.90 (+6% in California). There is a version of the software for just about everybody, so give us a call, let us know what kind of computer you've got, and we'll ship you a complete package. . . From Stock!

For those of you who are not yet familiar with the Micro Works Logo, we would like to introduce ourselves. We started out as 6800 users, and after spending more and more time looking for a variety of items that we felt would be real useful, we ended up building them ourselves. As a result, every Micro Works product has real Utility built right into it, along with our famous Quality. We buy only prime parts, direct from the best sources, and the P C boards are done for us by the finest house we can find. All of our units (except the DM-85, our first kit), are assembled, socketed, burned in and fully tested before shipment. As a result, we have had very few takers on our 90 day warranty. So take a look at our product line, and see if we can't help you out.

QUALITY HARDWARE FOR THE SS-50/30 BUS

PSB-08	8k of economical 2708 EPROM, 1k of high speed RAM, and the capability to move the I/O addresses to the top of memory, all for just	$119.95
B-08	Handy little 2708 Prom Burner, takes up only 1 I/O slot, complete with Textool Socket.	$99.95
DS-68	Our famous Digisector, hobby computing's most powerful and popular random access video digitizer, as originally designed for the SS-30 bus. Again, a single card unit.	$169.95
UIO	The Universal I/O board. A Proto board for the I/O bus, with a pre-wired interface chip, and room for more than 30 I.C.'s.	$24.95
X50/30	The only extender cards we've seen that have a ground plane on the front to reduce noise during debugging.	$29.95/$22.95
DM-85	Our first kit! A retro fit for later SSB Disk Controller boards that allows any combination of 8" and 5" drives.	$39.95

GENUINE 6800 SOFTWARE

MPRINT	All you need besides this 2708 is a pair of MP-LA's, and you can put a Malibu 160 high speed line printer on your system. (We'll sell you the printer, too.)	$39.95
BIO-PIC	The Computer Portrait and Biorhythm software package! 3k of EPROM, no source. Requires a DS-68, 16k of ram, and a Malibu Printer. Your own small Business for only	$175.00
U2708	The same program that's source listed in the B-08 Manual, burned into a 2708. Includes Block Move, Erase test, Burn, Verify, etc. Runs with all the bugs we know about.	$39.95
E6809	Brand New! An Emulator for the Motorola 6809. Allows you to test and debug '09 software before you get the chip. 3k, on disk. Specify Smoke Signal or Flex.	$49.95

Now, although you are going to start seeing ads for our new S-100 Digisector, the Micro Works is going to continue to develop innovative, useful products with the 6800/09 market in mind. Right now, we're working on a nifty graphics board; an 8k hunk of memory that displays either 256X256 by one dot, (great for games and graphs!) or 128X128 by 16 grey levels (Digisector compatable). We're also working on some interesting I/O units, and would be interested to hear from you regarding your needs. Please, don't ask us when they'll be available, because as soon as we know for SURE, we'll let you know. Right here, in Micro 68.

BANKAMERICARD

P.O. BOX 1110 DEL MAR, CA 92014
(714) 756-2687

master charge

We're not just blowing smoke

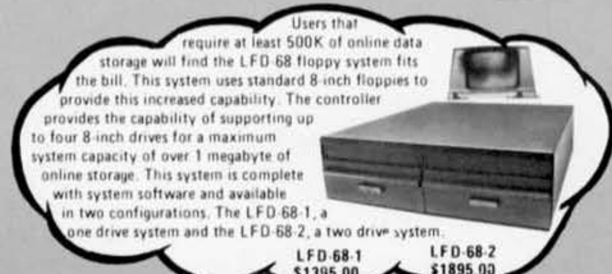
SMOKE SIGNAL BROADCASTING PRESENTS IT'S NEW...

8-INCH FLOPPY DISK SYSTEM

- SS-50 Bus Compatible
- Expandable to 1 Megabyte
- 500K Bytes of Online Storage
- Completely Software Compatible with existing BFD-68 Mini-Disk Systems

Users that require at least 500K of online data storage will find the LFD-68 floppy system fits the bill. This system uses standard 8-inch floppies to provide this increased capability. The controller provides the capability of supporting up to four 8-inch drives for a maximum system capacity of over 1 megabyte of online storage. This system is complete with system software and available in two configurations. The LFD-68-1, a one drive system and the LFD-68-2, a two drive system.

LFD-68-1 $1395.00 **LFD-68-2 $1895.00**



Ask about our new DFD-68-2 with two 8-inch double sided floppy disk drives with 512K bytes of storage per drive and capability of adding additional drives to provide over 2 megabytes of online storage

SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361, (213) 889-9340

SMOKE SIGNAL BROADCASTING®

31336 Via Colinas, Westlake Village, CA 91361
(213) 889-9340

☐ Send information on your LFD-68 Systems
☐ Send name of nearest dealer

Name ______
Address ______
Company ______
City ______
State/Zip ______

'68' Micro Journal
3018 Hamill Rd.
Hixson, TN 37343

Application to Mail at Second Class Postage Rate is Pending at Chattanooga, TN.

TIME VALUE
PLEASE DO NOT DELAY